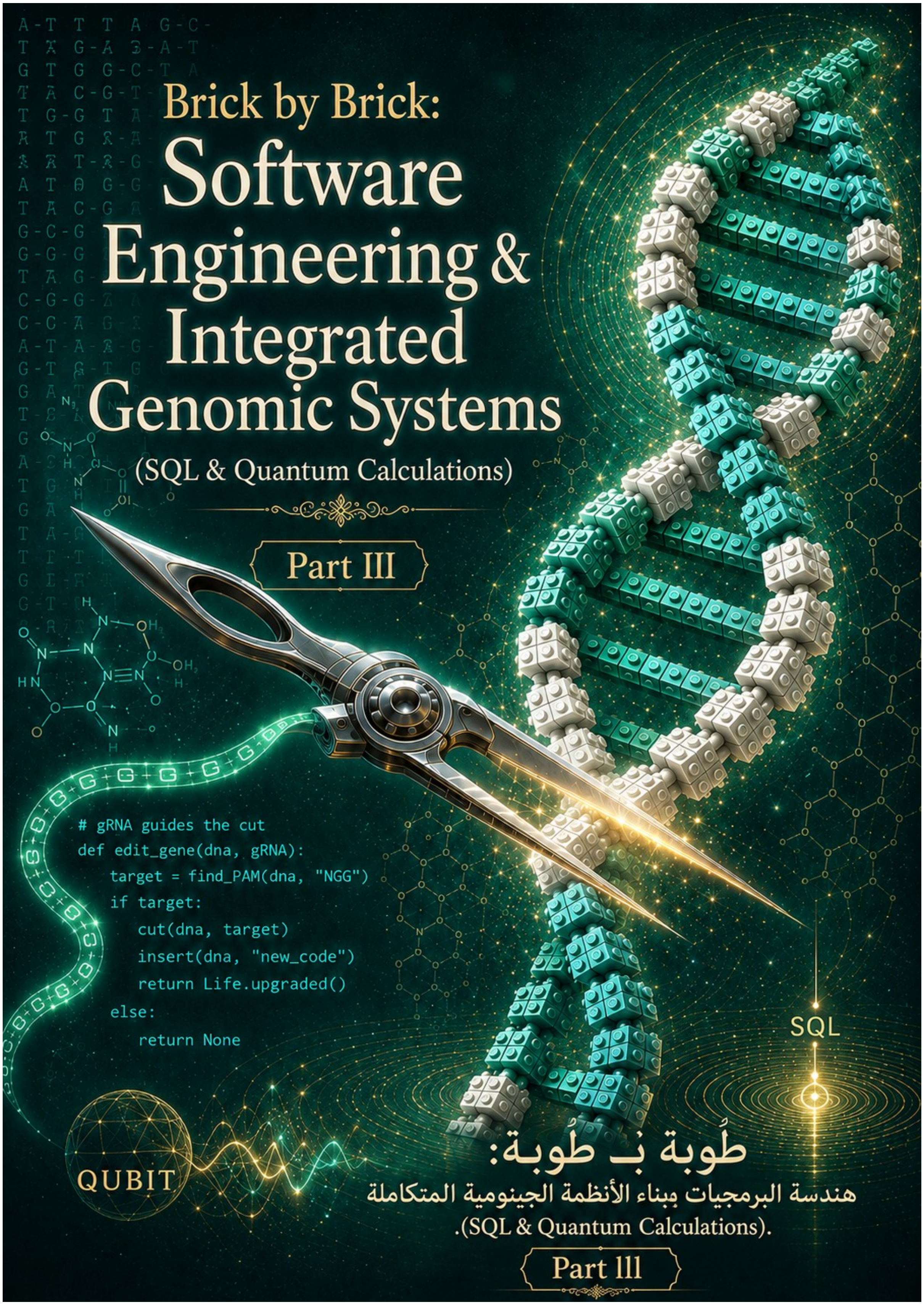
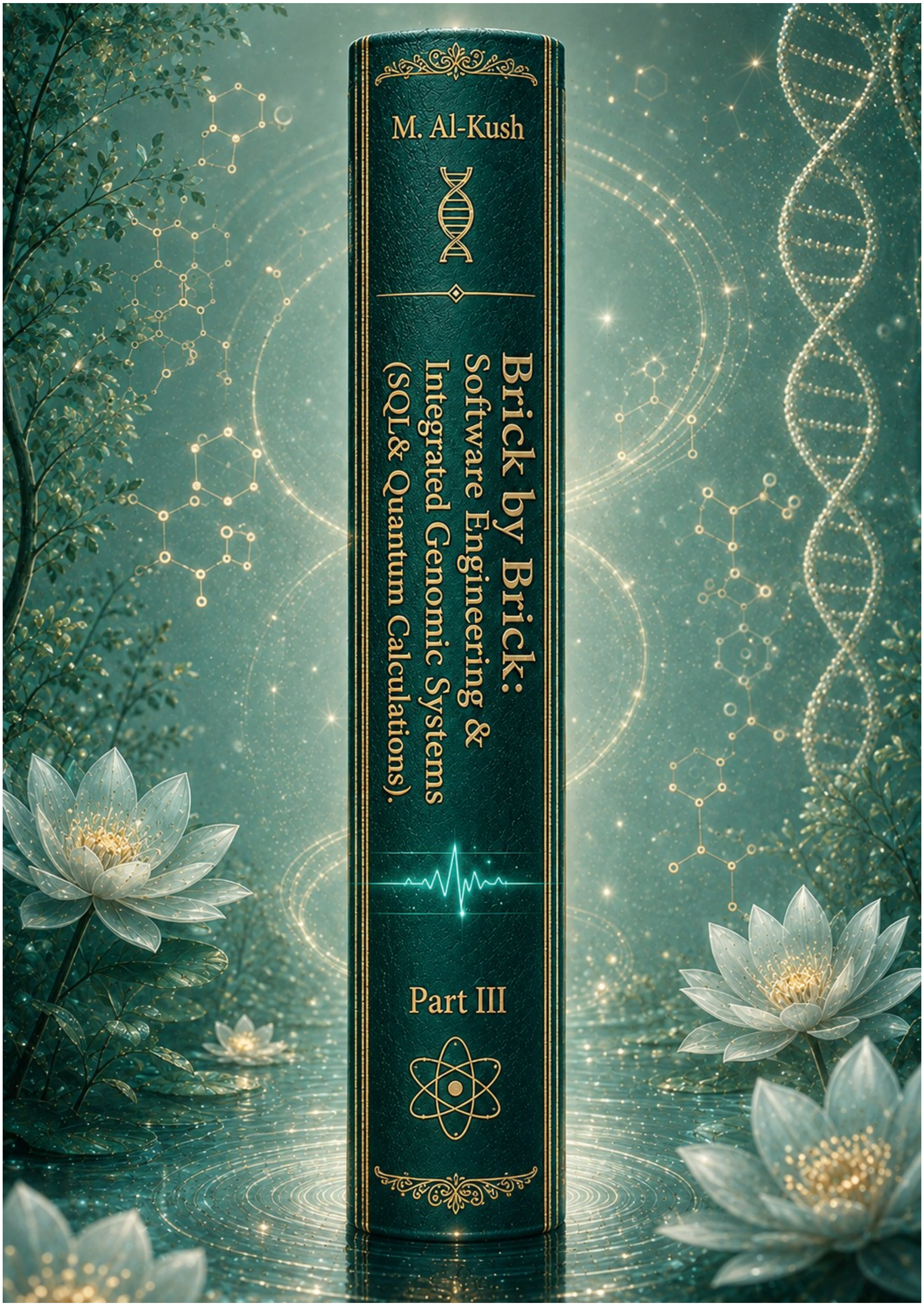
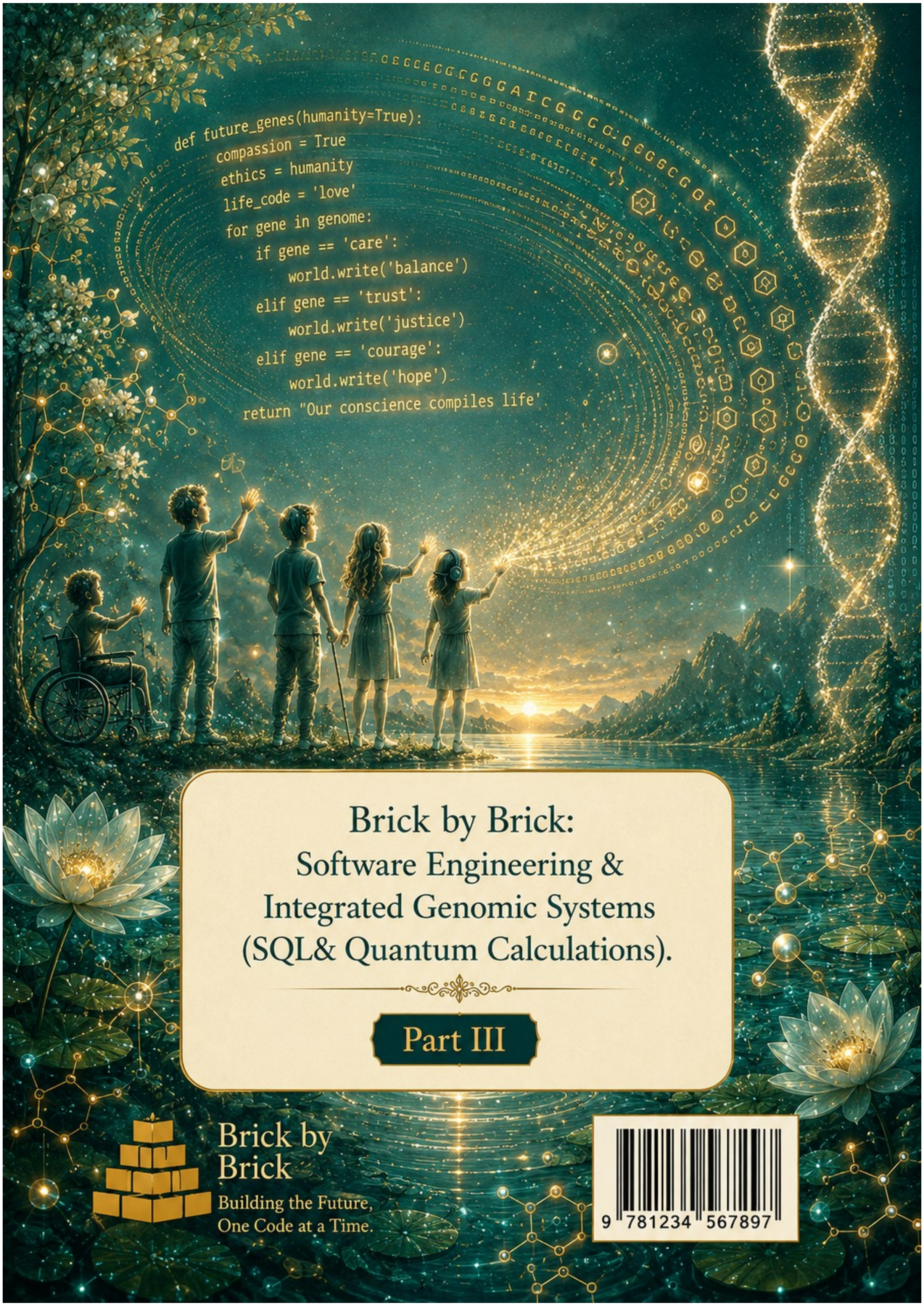

Brick by Brick:
Software Engineering & Integrated Genomic Systems
(SQL & Quantum Calculations)
Part III
gRNA guides the cut
def edit_gene(dna, gRNA):
target = find_PAM(dna, "NGG")
if target:
cut(dna, target)
insert(dna, "new_code")
return Life.upgraded()
else:
return None
SQL
QUBIT
طُوبة بـ طُوبة:
هندسة البرمجيات ببناء الأنظمة الجينومية المتكاملة
.(SQL & Quantum Calculations)
Part III


M. Al-Kush
Brick by Brick:
Software Engineering &
Integrated Genomic Systems
(SQL& Quantum Calculations).
Part III


def future_genes(humanity=True):
compassion = True
ethics = humanity
life_code = 'love'
for gene in genome:
if gene == 'care':
world.write('balance')
elif gene == 'trust':
world.write('justice')
elif gene == 'courage':
world.write('hope')
return "Our conscience compiles life'
Brick by Brick:
Software Engineering &
Integrated Genomic Systems
(SQL& Quantum Calculations).
Part III
Brick by Brick
Building the Future, One Code at a Time.
9 781234 567897

طوبةٌ بـ طوبة:

هندسة البرمجيات وبناء الأنظمة الجينومية المتكاملة (SQL & Quantum Calculations).

Brick by Brick: Software Engineering & Integrated Genomic Systems (SQL & Quantum Calculations).

Part III

Copyright and Licensing

Email: mohamedelzoghpe905@gmail.com.

الفهرس

تمهيد

Table of Contents

في الجزء الأول، بدأت رحلتك مع لغة الحياة ذاتها: DNA وRNA والبروتين. تعلمت كيف تتحول المفاهيم الجزيئية المجردة إلى تمثيلات رقمية يمكن للحاسوب معالجتها. ثم خطوت إلى عالم Biopython، حيث تعرفت على Seq و SeqRecord و SeqFeature و SeqIO، وتعلمت كيف تتنقل بين صيغ الملفات الأساسية: FASTA و FASTQ و GenBank. تعمقت في الخوارزميات، واستكشفت CRISPR من منظور علمي وحاسوبي آمن، وبنيت الحزمة التعليمية biobrick_security. ثم ارتقيت إلى تحليل بيانات التعبير الجيني (RNA-seq)، والإحصاء الحيوي، والبحث في المصادر العلمية، وصولاً إلى المشروع المتكامل الذي جمع كل هذه الخيوط في نسيج واحد.

في هذه المرحلة من المسار، وضعت حجر الزاوية للتحول البرمجي الحقيقي: Python، البيئات الافتراضية، إدارة الحزم، التعمق في Biopython، ثم البرمجة كائنية التوجه (OOP). لم تكن هذه مجرد أدوات جديدة؛ كانت إعادة تنظيم لعقلك البرمجي. تعلمت أن تنظر إلى البيانات والسلوك ككيان واحد، وأن تصمم وحدات قابلة

لإعادة الاستخدام والاختبار. لقد أتقنت كيفية بناء كائن SeqRecord الذي يحمل في داخله التسلسل، والهوية، والبيانات الوصفية، والميزات.

الآن، مع الجزء الثالث، يتحول المبرمج إلى مهندس. الكائنات التي بنيتها في الجزء الثاني، مثل Seq و SeqRecord، كانت تعيش في الذاكرة العشوائية (RAM). طالما كان البرنامج يعمل، كانت الكائنات موجودة. لكن بمجرد أن ينتهي البرنامج، تتبخر كل تلك الكائنات وكأنها لم تكن. السؤال الهندسي الأول هو: كيف نجعل هذه الكائنات دائمة (Persistent)؟ كيف نخزنها بحيث يمكننا استعادتها بعد أيام أو سنوات، دون أن نفقد العلاقات المعقدة بينها؟

هنا تظهر الحاجة إلى قواعد البيانات العلائقية (Relational Databases) ولغة SQL. لكننا لن نبدأ بتعلم أوامر SQL عن ظهر قلب. بل سنبدأ بالسؤال الأهم: كيف نفكر في البيانات

كمنظومة متكاملة؟ وكيف ننقل تفكيرنا الكائني (OOP) من الذاكرة المؤقتة إلى التخزين الدائم؟

التدرج الهندسي لهذا الجزء واضح وممنهج: يبدأ بتخزين الكائنات في قاعدة بيانات، ثم يوحد بيئة التشغيل باستخدام الحاويات، ثم ينظم سير العمل، ثم يختبر وينشر الحزمة البرمجية، ثم يبني واجهة للمستخدم، ثم يتعامل مع البيانات الضخمة، وأخيراً يدمج كل شيء في مشروع متكامل. كل خطوة تحل مشكلة ناشئة عن الخطوة التي سبقتها.

الباب الأول: من الملفات إلى قواعد البيانات

Unit One: From Files to Databases

لوحة توضيحية: الحلقة الهجينة الكلاسيكية–الكمومية

لوحة توضيحية: الهاميلتوني والطاقة

لوحة توضيحية: تمثيل الحالة الكمومية

In Part One, you began your journey with the language of life itself: DNA, RNA, and protein. You learned how abstract molecular concepts transform into digital representations that computers can process. Then you stepped into the world of Biopython, where you encountered Seq, SeqRecord, SeqFeature, and SeqIO, and learned to navigate essential file formats: FASTA, FASTQ, and GenBank. You delved into algorithms, explored CRISPR from a scientific and secure computational perspective, and built the educational biobrick_security package. You then advanced to gene expression analysis (RNA-seq), biostatistics, scientific literature searching, and finally the integrated project that wove all these threads into a single fabric.

Part Two laid the cornerstone of true software transformation: Python, virtual environments, package management, deep Biopython, and Object-Oriented Programming (OOP). These were not merely new tools; they were a

reorganisation of your programming mindset. You learned to see data and behavior as a unified entity, and to design reusable, testable modules. You mastered how to build a SeqRecord object that encapsulates the sequence, identity, metadata, and features within a single, coherent structure.

لوحة توضيحية: خط أنابيب الذكاء الاصطناعي الجينومي

Now, with Part Three, the programmer transforms into an engineer. The objects you built in Part Two, such as Seq and SeqRecord, lived in Random Access Memory (RAM). As long as the program ran, the objects existed. But once the program ended, all those objects vanished as if they never were. The first engineering question is: How do we make these objects persistent? How do we store them so we can retrieve them days or years later, without losing the complex relationships between them?

This is where Relational Databases and SQL enter the picture. However, we will not start by memorizing SQL commands. Instead, we will begin with the most important question: How do we think about data as an integrated system? And how do we transfer our Object-Oriented thinking from transient memory to permanent storage?

The engineering progression of this part is clear and methodical: it begins with storing objects in a database, then unifies the runtime environment using containers, then organizes the workflow, then tests and distributes the software package, then builds a user interface, then handles large-scale data, and finally integrates everything into a capstone project. Each step solves a problem created by the preceding step.

Educational plate: The Classical–Quantum Hybrid Loop

لوحة توضيحية: BLOSUM كمصفوفة

Educational plate: Hamiltonian and Energy

لوحة توضيحية: جسر الجبر الخطي

Educational plate: Quantum State Representation

لوحة توضيحية: الدقة العددية

Educational plate: The AI Genomic Pipeline

لوحة توضيحية: المعاملات وسلامة البيانات

Educational plate: BLOSUM as a Matrix

Educational plate: The Linear Algebra Bridge

لوحة توضيحية: مقايضات الفهرسة

Educational plate: Numerical Precision

لوحة توضيحية: المعالجة المتدفقة على نطاق واسع

Educational plate: Transactions and Integrity

لوحة توضيحية: طبقة المستودعات OOP

Educational plate: Indexing Trade-offs

لوحة توضيحية: خط أنابيب الاستعلام SQL

Educational plate: Streaming at Scale

لوحة توضيحية: سُلّم التطبيع

Educational plate: The OOP Repository Layer

لوحة توضيحية: التفكير بالكيانات والعلاقات

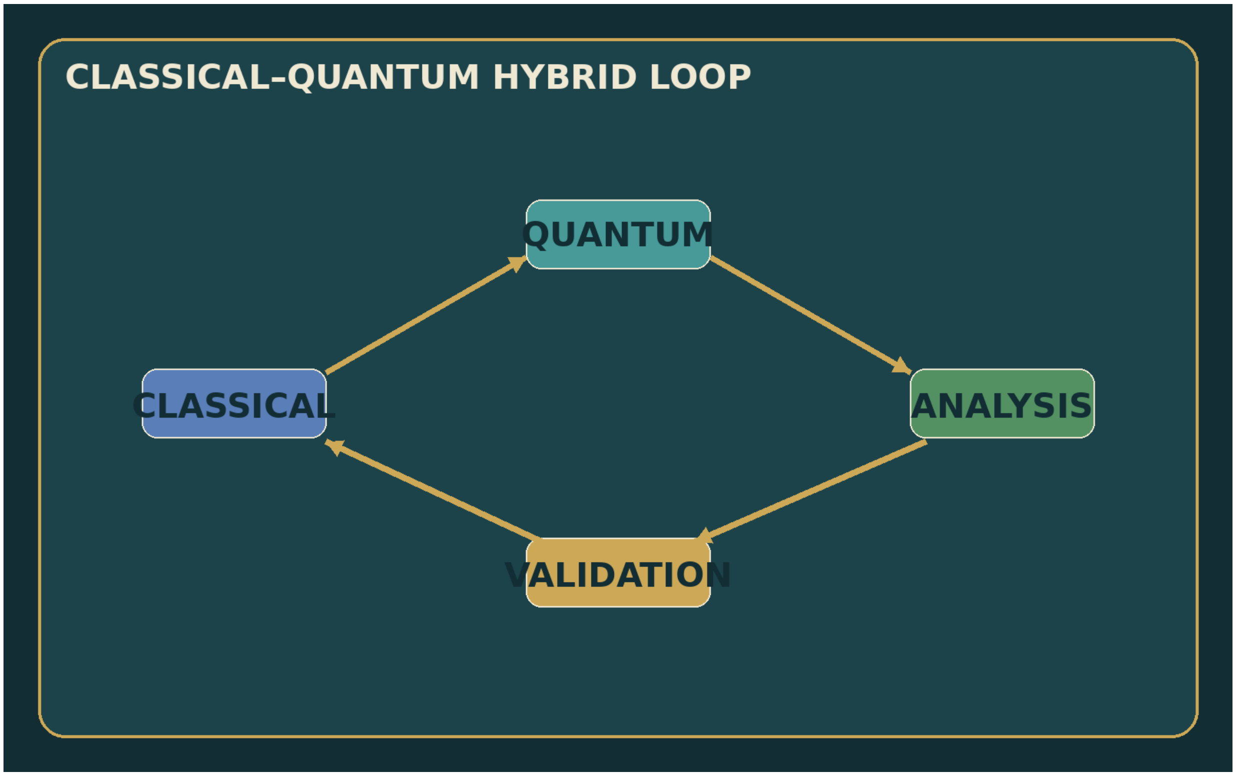
Educational plate: The SQL Query Pipeline

لوحة توضيحية: من الملفات إلى قواعد البيانات

Educational plate: The Normalization Ladder

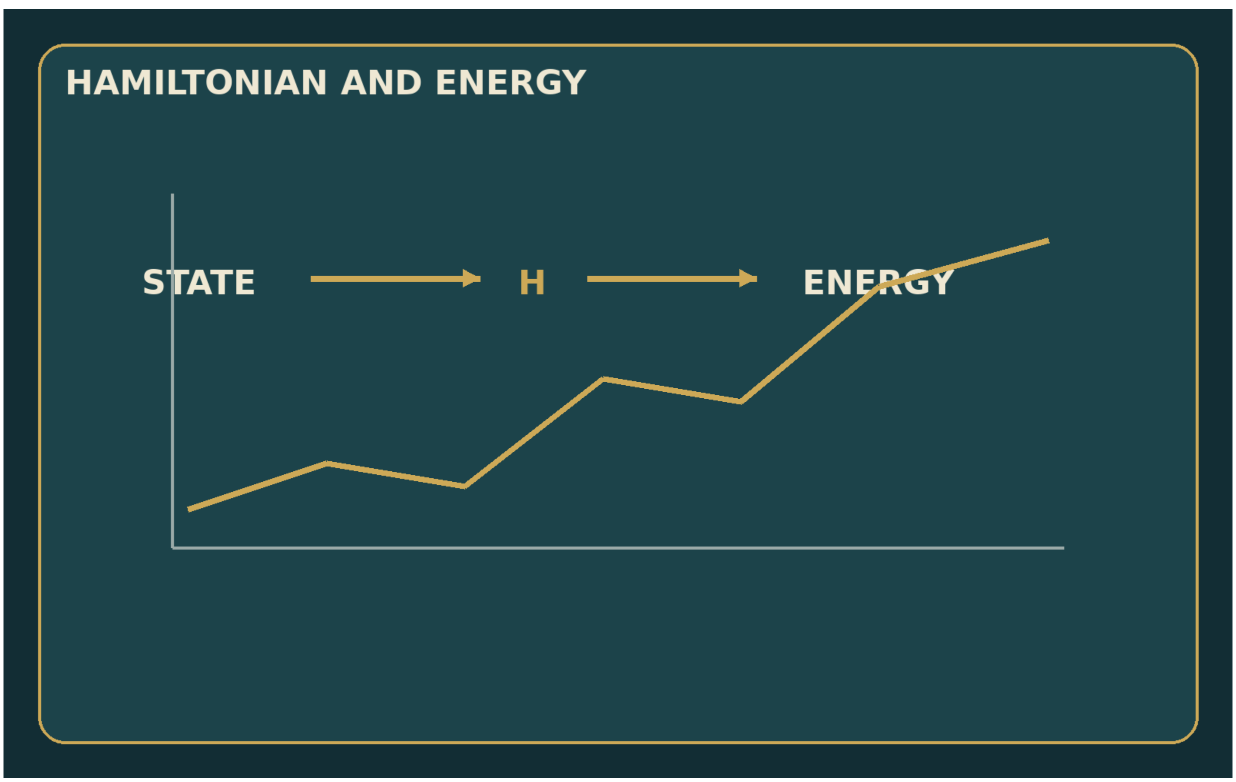
Educational plate: Entity–Relationship Thinking

Educational plate: From Files to Databases



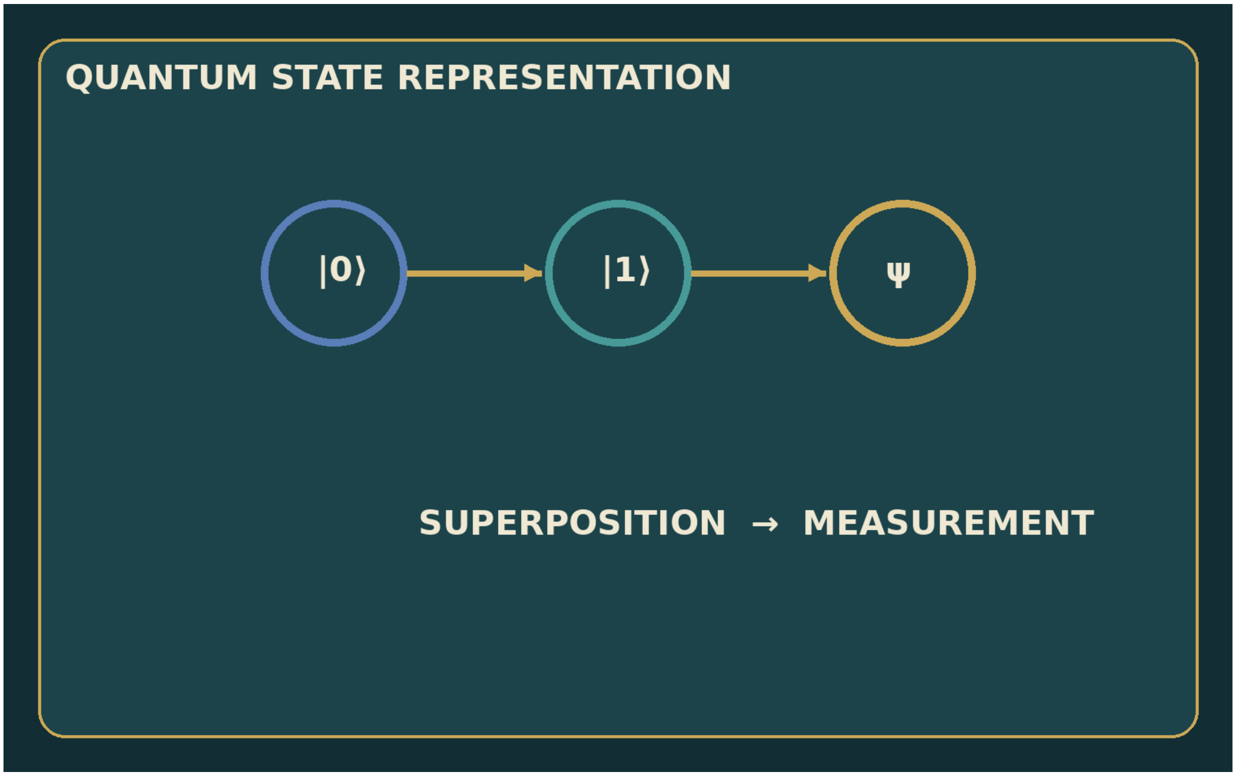
الحلقة الهجينة الكلاسيكية–الكمومية

CLASSICAL–QUANTUM HYBRID LOOP



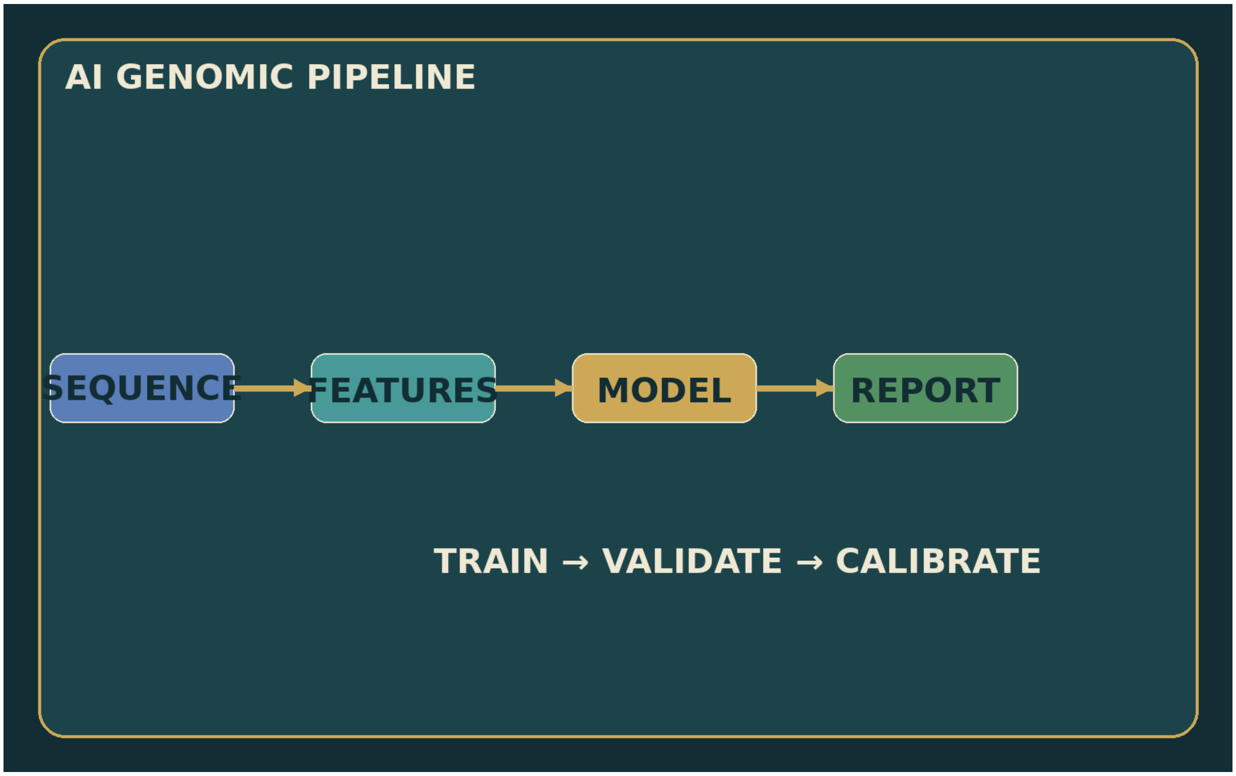
الهاميلتوني والطاقة

HAMILTONIAN AND ENERGY



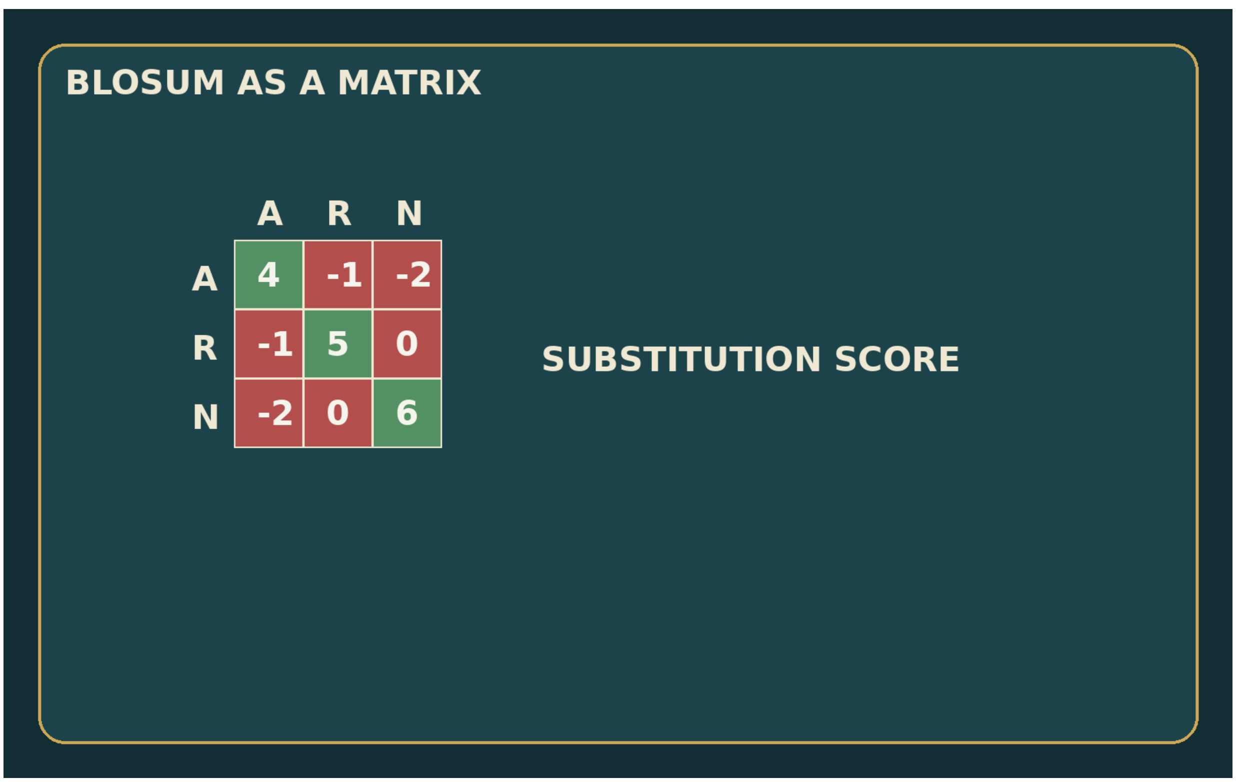
تمثيل الحالة الكمومية

QUANTUM STATE REPRESENTATION



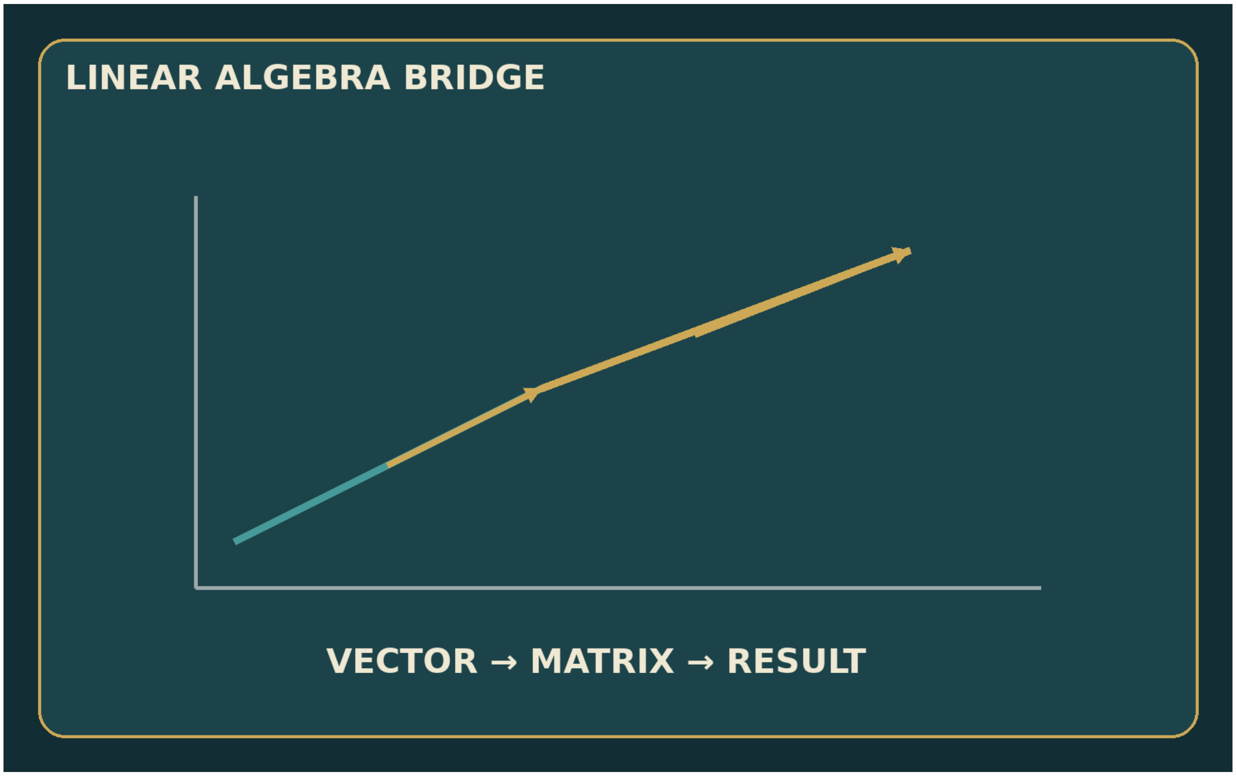
خط أنابيب الذكاء الاصطناعي الجينومي

AI GENOMIC PIPELINE



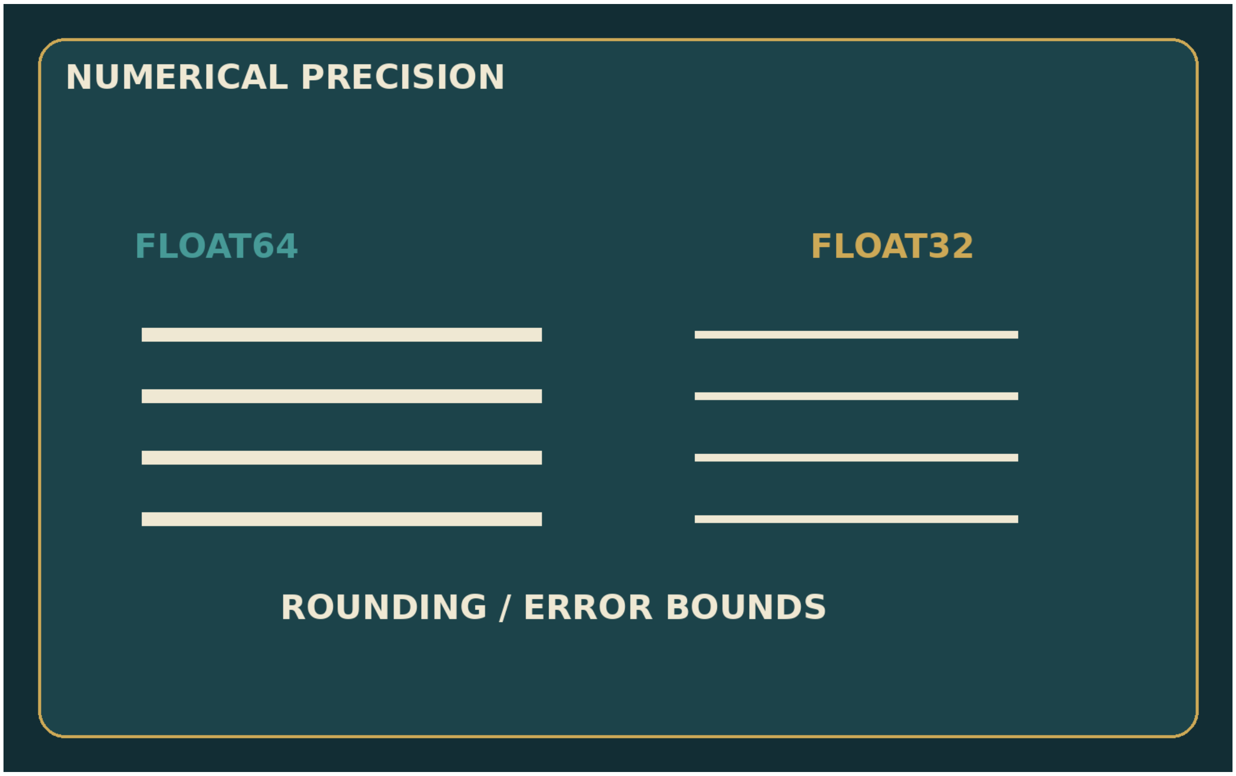
BLOSUM كمصفوفة

BLOSUM AS A MATRIX



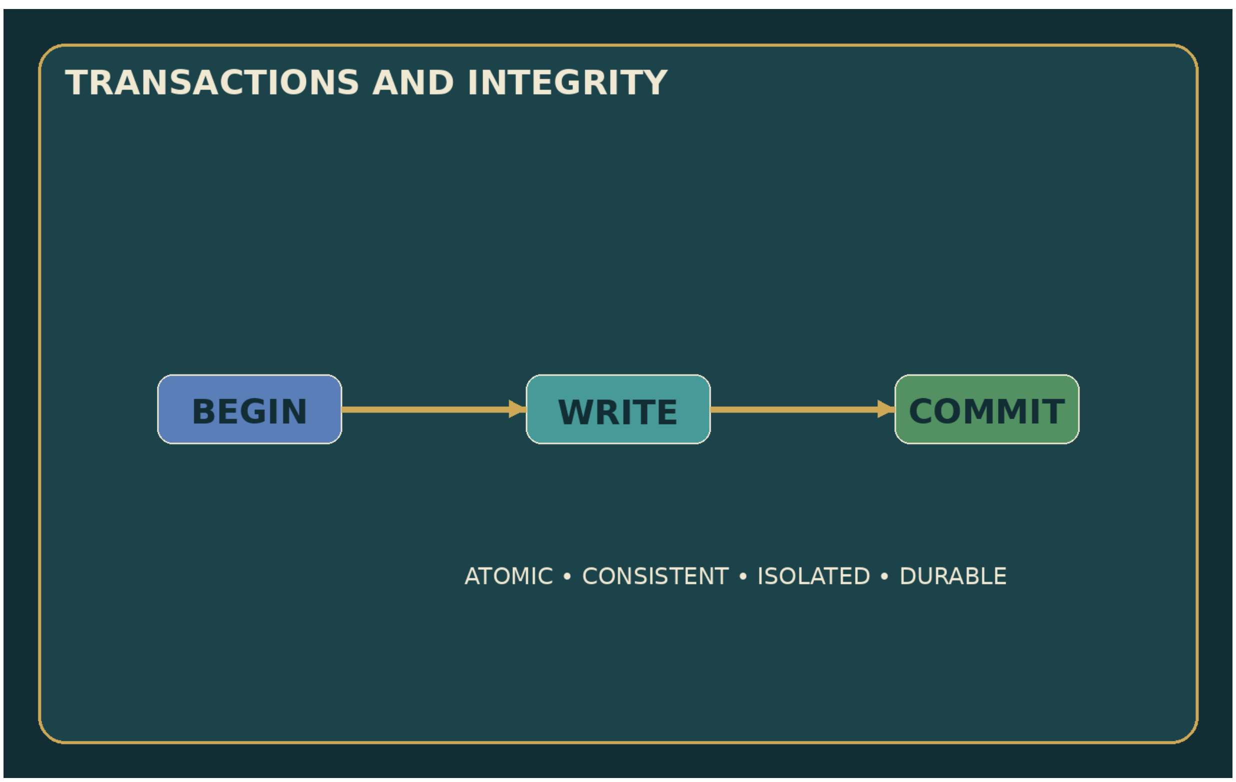
جسر الجبر الخطي

LINEAR ALGEBRA BRIDGE



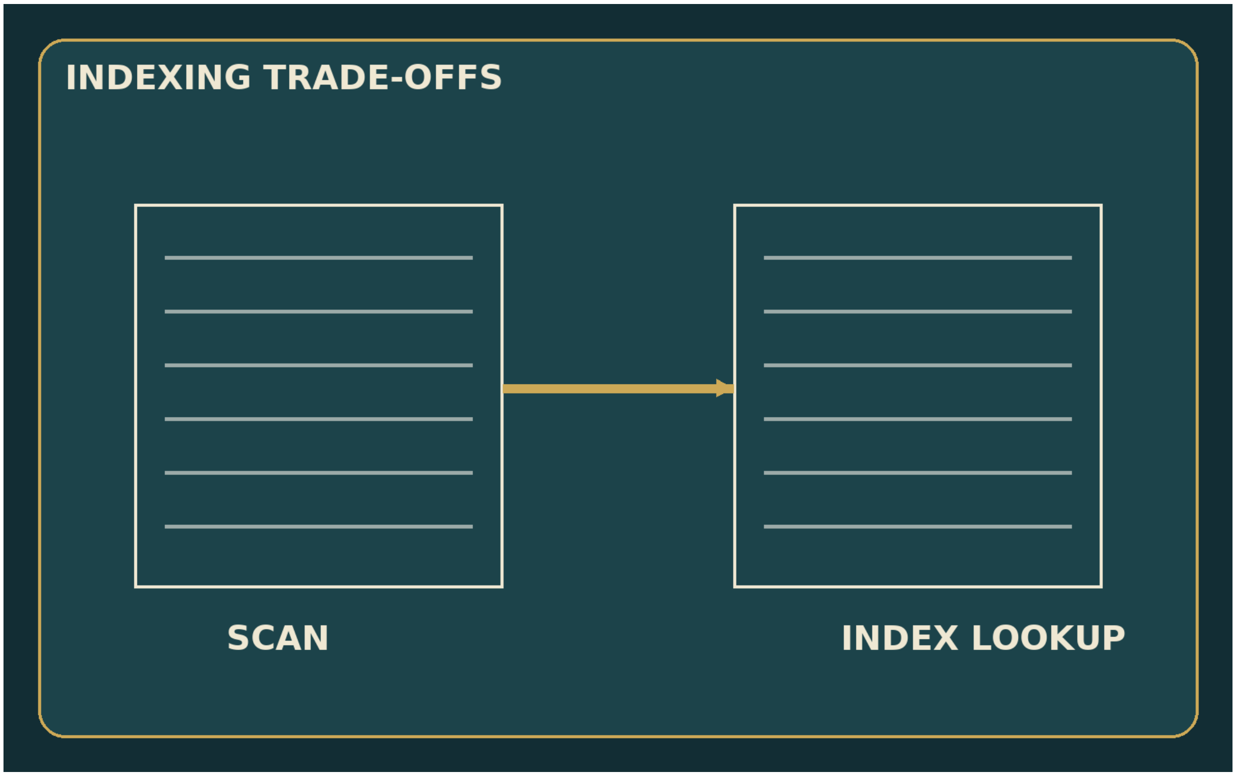
الدقة العددية

NUMERICAL PRECISION



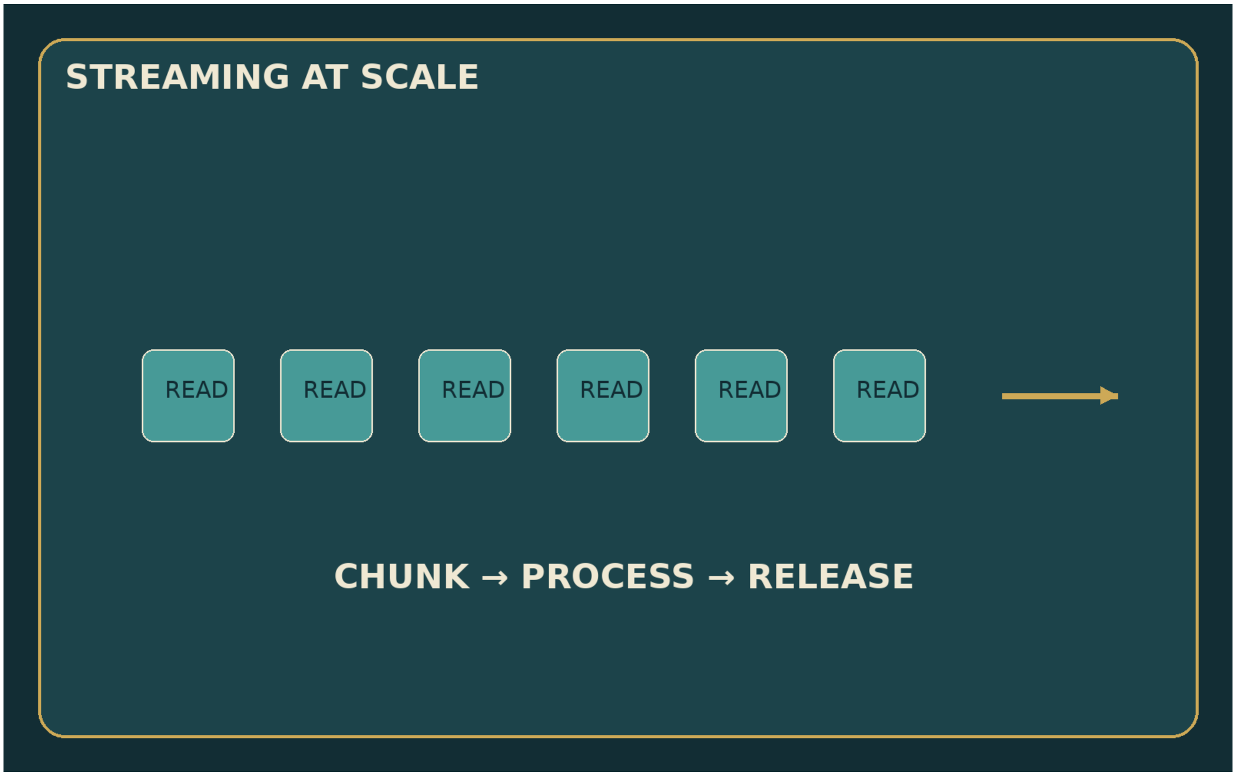
المعاملات وسلامة البيانات

TRANSACTIONS AND INTEGRITY



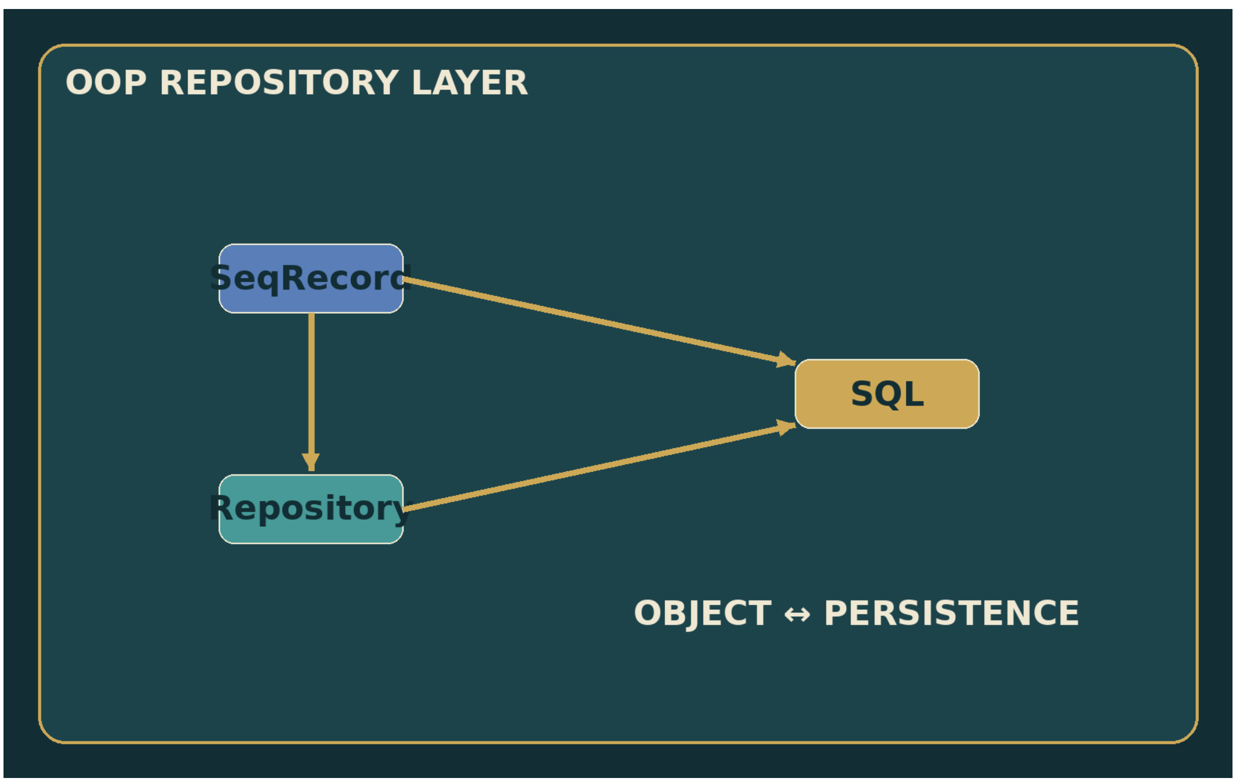
مقايضات الفهرسة

INDEXING TRADE-OFFS



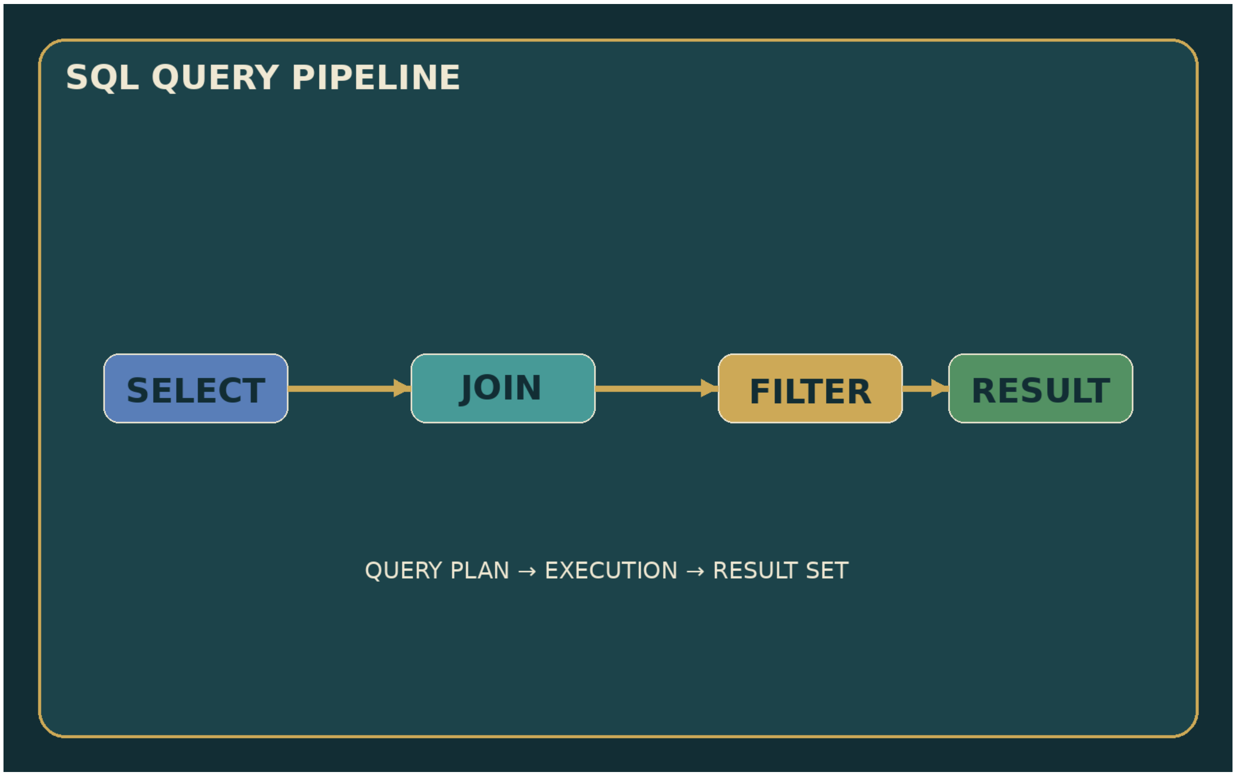
المعالجة المتدفقة على نطاق واسع

STREAMING AT SCALE



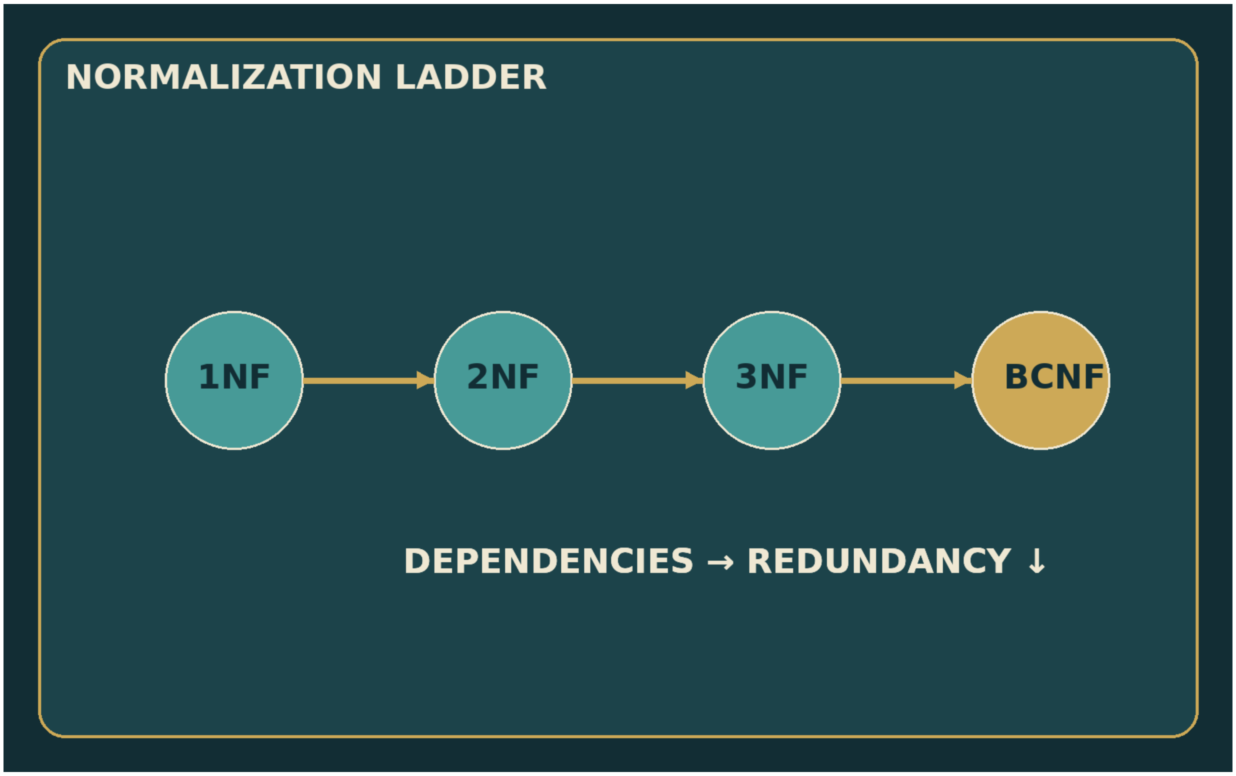
طبقة المستودعات **OOP**

OOP REPOSITORY LAYER



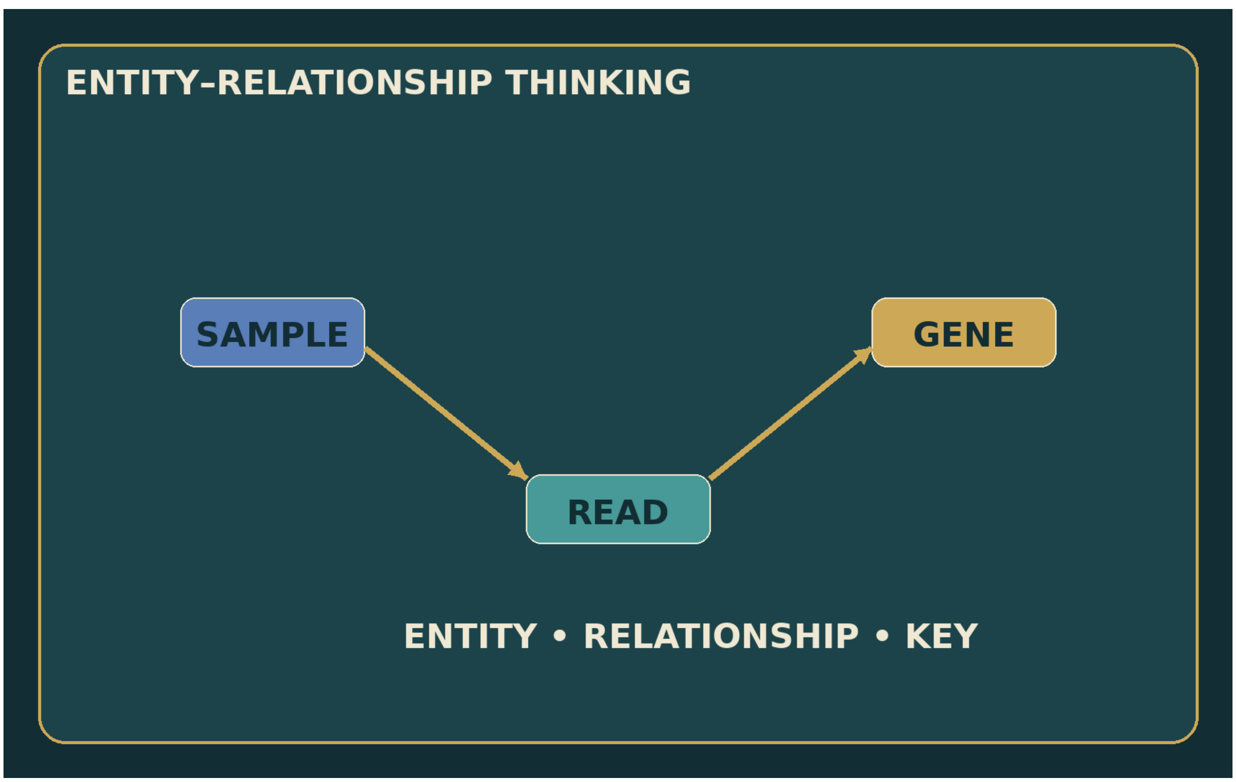
خط أنابيب الاستعلام **SQL**

SQL QUERY PIPELINE



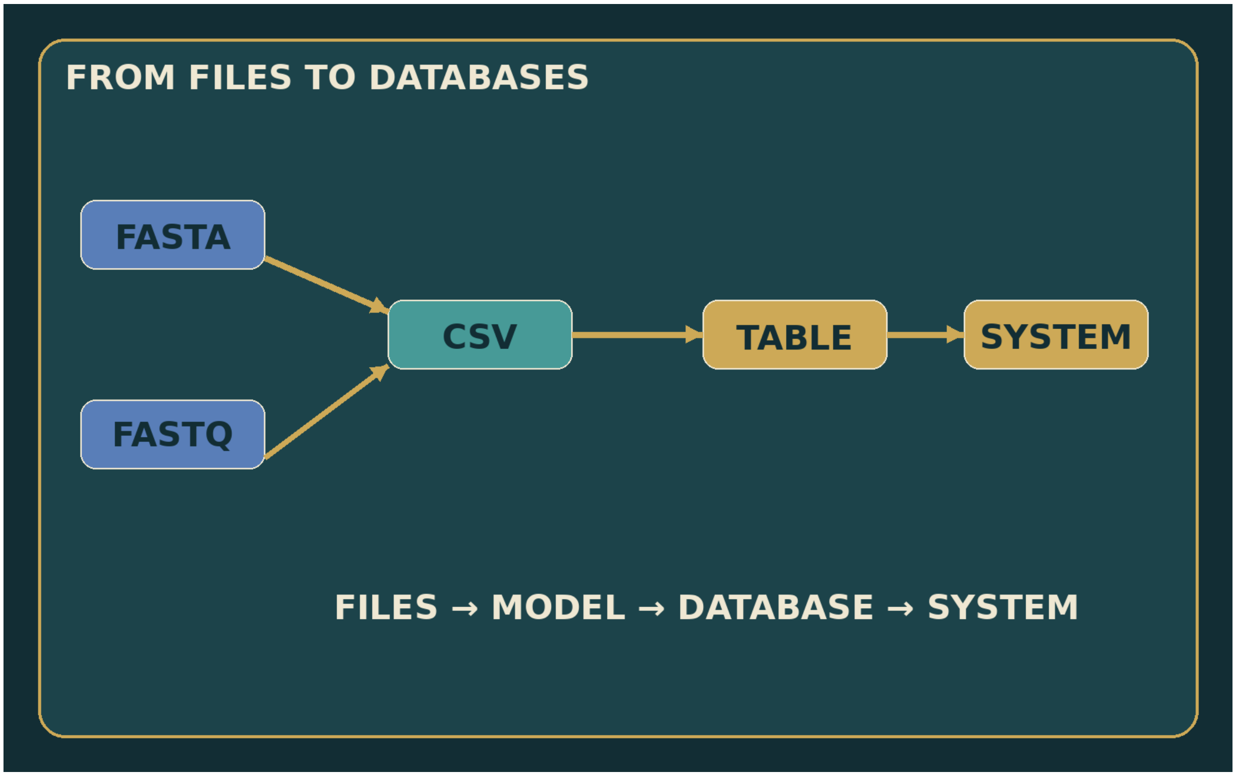
سُلّم التطبيع

NORMALIZATION LADDER



التفكير بالكيانات والعلاقات

ENTITY–RELATIONSHIP THINKING



من الملفات إلى قواعد البيانات

FROM FILES TO DATABASES

الغرض من هذا الباب:

تحويل التفكير من كائنات عابرة في الذاكرة إلى سجلات دائمة في قاعدة بيانات. إدراك أن الملفات المسطحة، رغم قوتها في التبادل، تفشل هندسياً عندما تصبح البيانات مترابطة

ومعقدة. بناء جسر منهجي بين SeqRecord في Biopython و Row في SQLite، مع إرساء مبدأ أن كل كائن في OOP له نظير في جدول، وكل علاقة بين الكائنات لها نظير في مفتاح أجنبي.

Purpose of this unit:

To transform thinking from transient objects in memory to permanent records in a database. To realize that flat files, despite their power in exchange, fail structurally when data becomes interconnected and complex. To build a methodological bridge between SeqRecord in Biopython and Row in SQLite, establishing the principle that every object in OOP has a counterpart in a table, and every relationship between objects has a counterpart in a foreign key.

الفصل الأول: لماذا لا تكفي الملفات؟ — القيود الهندسية للتخزين المسطح

Chapter One: Why Files Are No Longer Enough — The Engineering Limitations of Flat Storage

هناك فرق هندسي مهم بين «تنسيق تبادل» و«نظام إدارة بيانات». يمكن أن يكون FASTA أو FASTQ ممتازًا لنقل التسلسل، لكنه لا يقدّم بمفرده هوية علائقية، ولا قيودًا تمنع التناقض، ولا سجلًا واضحًا للعلاقات بين العينة والتجربة والمرجع والنتيجة. لذلك لا ينبغي أن نضع الملف وقاعدة البيانات في منافسة مباشرة؛ الملف يظل واجهة تبادل وأرشفة مفيدة، بينما تتولى قاعدة البيانات إدارة العلاقات والاتساق والاستعلامات القابلة للتدقيق. هذه التفرقة تمنع خطأً شائعًا: محاولة تحويل كل شيء إلى صف واحد ضخم لمجرد أن النص الأصلي كان ملفًا.

An important engineering distinction is the difference between an “exchange format” and a “data-management system.” FASTA or FASTQ can be excellent for transporting sequence data, but by themselves they do not provide relational identity, integrity constraints, or an explicit record of the relationships among a sample, an experiment, a reference, and a result. We therefore should not treat files and databases as direct competitors: files remain useful for exchange and archival, while the database manages

relationships, consistency, and auditable queries. This distinction prevents a common mistake: forcing everything into one enormous row merely because the original source was a file.

الشكل 1 — الخريطة المفاهيمية للفصل

Figure 1 — Conceptual map of the chapter

1.5 FASTA وFASTQ وGenBank — لماذا تمثل الملفات بداية الطريق فقط؟

الشكل 1 — سير العمل الهندسي للفصل

Figure 1 — Engineering workflow of the chapter

عند الانتقال من الجزء السابق إلى هذا الفصل، أصبحت قادرًا على قراءة ملفات التسلسل باستخدام Biopython. هنا ستستخدم المهارة نفسها لاستخراج المشكلة الهندسية: الملف مناسب للتبادل والقراءة المتتابعة، لكنه لا يمنحك وحده آلية علائقية للبحث عن جميع العينات أو ربطها بتجارب ونسخ وملاحظات متعددة.

1.5 FASTA, FASTQ, and GenBank — Why Files Are Only the Beginning.

At this stage of the path, you learned to read sequence files with Biopython. Here you will use the same skill to expose the engineering problem: a file is excellent for exchange and sequential reading, but by itself it does not provide a relational mechanism for connecting many samples, experiments, versions, and observations.

Python

```
from Bio import SeqIO


for record in SeqIO.parse("example.fasta", "fasta"):
    print(record.id)
    print(len(record.seq))
```

Python

```
from Bio import SeqIO


for record in SeqIO.parse("example.gb", "genbank"):
    print(record.id)
    for feature in record.features:
        print(feature.type, feature.location)
```

SeqRecord وSeqFeature يجعلان البيانات الغنية قابلة للمعالجة داخل Python، لكن عندما تحتاج إلى سؤال مثل «أعطني كل العينات التي تحتوي على طفرة معينة مع إصدار المرجع الذي استُخدم» تصبح قاعدة البيانات أكثر ملاءمة.

SeqRecord and SeqFeature make richly annotated data convenient inside Python, but when you need a question such as “return every sample containing a specific mutation together with the reference version used,” a database becomes a more appropriate representation.

المعرفة التي بُنيت في الجزء الثاني تقودنا هنا إلى أن، بلغت مرحلة متقدمة من التحليل البرمجي. كنت تقرأ ملفات FASTA التي تحتوي على تسلسلات DNA، وتستخرج البيانات الوصفية من سطور الرأس، وتحلل ملفات FASTQ التي تضيف معلومات الجودة لكل قاعدة، وتتعامل مع ملفات GenBank التي تحمل بنية شبه مهيكلة غنية بالميزات والتعليقات. كما تعلمت قراءة ملفات CSV التي تخزن جداول التعبير الجيني، وربما تصدير النتائج إلى JSON للتواصل بين الأنظمة المختلفة. في كل هذه الحالات، كنت تتعامل مع "ملف" — كتلة من البايتات على القرص الصلب، لها بداية ونهاية، وتُقرأ بالتسلسل.

As you move from the previous part into this chapter, you reached an advanced stage of programmatic analysis. You were reading FASTA files containing DNA sequences, extracting metadata from header lines, analyzing FASTQ files that add quality information for each base, and handling GenBank files with a rich, semi-structured layout of features and annotations. You also learned to read CSV files storing gene expression tables, and perhaps export results to JSON for inter-system communication. In all these cases, you were dealing with a "file"—a block of bytes on the hard drive, with a beginning and an end, read sequentially.

هذه المهارات ضرورية ولا غنى عنها. الملفات هي لغة التبادل الأساسية في المعلوماتية الحيوية؛ فهي تنتقل بين المختبرات، وتُرفع إلى قواعد البيانات العامة، وتُشارك مع

الزملاء، وتُخزن كأرشيف للنتائج. لكن عندما تنتقل من مرحلة "تحليل مشروع صغير" إلى مرحلة "بناء نظام متكامل"، تبدأ الملفات في التصدع تحت وطأة التعقيد. المشكلة ليست في الملف نفسه، بل في غياب البنية التي تنظم العلاقات بين الملفات المختلفة، وتضمن اتساقها، وتمنع تكرارها، وتجعل البحث فيها فعالاً.

These skills are essential and indispensable. Files are the primary language of exchange in bioinformatics; they travel between laboratories, are uploaded to public databases, shared with colleagues, and stored as result archives. However, when you move from the "small project analysis" phase to the "building an integrated system" phase, files begin to crack under the weight of complexity. The problem is not the file itself, but the absence of a structure that organizes relationships between different files, ensures their consistency, prevents their duplication, and makes searching within them efficient.

دعنا نحلل ثلاث مشاكل هندسية جوهرية تجعل الملفات غير كافية كنظام لإدارة البيانات الجينومية:

Let us analyze three fundamental engineering problems that make files insufficient as a system for managing genomic data:

المشكلة الأولى: التكرار (Redundancy) والاعتماد الوظيفي (Functional Dependency)

Problem One: Redundancy and Functional Dependency.

تخيل أن لديك ملف CSV يحتوي على بيانات العينات (Sample Metadata). في هذا الملف، كل صف يمثل عينة واحدة. الصفوف تحتوي على: sample_id، patient_id، patient_age، collection_date، tissue_type، patient_sex. إذا كان لديك مريض واحد

له عشر عينات مختلفة (مأخوذة في أوقات مختلفة أو من أنسجة مختلفة)، فإنك ستكرر patient_age و patient_sex في كل صف من صفوف العشر عينات. هذا التكرار يبدو غير ضار في البداية، لكنه يتحول إلى كارثة صيانة عندما يكتشف خطأ في عمر المريض، أو عندما يتغير في الواقع. ستحتاج إلى تعديل عشر صفوف بدلاً من صف واحد. وإذا كان لديك مئات المرضى وآلاف العينات، يصبح التعديل اليدوي مستحيلاً، والبرمجي يصبح معقداً وعرضة للخطأ.

Imagine you have a CSV file containing sample metadata. In this file, each row represents one sample. The rows contain: sample_id, patient_id, patient_age, patient_sex, tissue_type, collection_date. If one patient has ten different samples (taken at different times or from different tissues), you will repeat patient_age and patient_sex in every one of the ten sample rows. This redundancy seems harmless at first, but it becomes a maintenance catastrophe when an error is discovered in the patient's age, or when it

changes in reality. You will need to modify ten rows instead of one. And if you have hundreds of patients and thousands of samples, manual modification becomes impossible, and programmatic modification becomes complex and error-prone.

في نظرية قواعد البيانات، تُعرف هذه المشكلة باسم "الاعتماد الوظيفي" (Functional Dependency). عمود patient_age يعتمد وظيفياً على patient_id، وليس على sample_id. بتعبير آخر، عمر المريض هو صفة للمريض، وليست صفة للعينة. بتخزينها في جدول العينات، فإننا نخالف مبدأ التطبيع (Normalization) الأولي، ونخلق "تكراراً" يهدد سلامة البيانات.

In database theory, this problem is known as "Functional Dependency". The patient_age column functionally depends on patient_id, not on sample_id. In other words, the patient's age is an attribute of the patient, not of the sample. By storing it in the samples table, we violate the initial normalization principle, creating "redundancy" that threatens data integrity.

في الممارسة العملية، هذا التكرار يؤدي إلى:

1. إهدار المساحة: البيانات المكررة تستهلك مساحة تخزين، خصوصاً مع ملايين السجلات.
2. تباطؤ الاستعلامات: تكرار البيانات يزيد من حجم الجداول، مما يبطئ عمليات القراءة والبحث.
3. شذوذ التحديث (Update Anomaly): تغيير قيمة واحدة يتطلب تحديث عدة صفوف، وإلا أصبحت البيانات غير متسقة.
4. شذوذ الإدخال (Insertion Anomaly): لا يمكن إدخال مريض جديد دون أن يكون له عينة على الأقل، لأن البيانات مقيدة بجدول العينات.
5. شذوذ الحذف (Deletion Anomaly): حذف آخر عينة لمريض يعني فقدان جميع بياناته.

In practice, this redundancy leads to:

1. Space waste: Duplicated data consumes storage space, especially with millions of records.
2. Slower queries: Data duplication increases table size, slowing down read and search operations.
3. Update Anomaly: Changing one value requires updating multiple rows, otherwise data becomes inconsistent.
4. Insertion Anomaly: A new patient cannot be entered without having at least one sample, because data is tied to the samples table.
5. Deletion Anomaly: Deleting the last sample of a patient means losing all their data.

الحل الهندسي لهذه المشكلة هو فصل الكيانات: نضع بيانات المريض في جدول Patients، وبيانات العينة في جدول Samples، ونربطهما بمفتاح أجنبي (patient_id). بهذه الطريقة، يتغير عمر المريض في مكان واحد فقط، وجميع العينات تشير إليه تلقائياً.

The engineering solution to this problem is Entity Separation: we put patient data in a Patients table, sample data in a Samples table, and link them with a foreign key (patient_id). This way, the patient's age changes in only one place, and all samples automatically refer to it.

المشكلة الثانية: الاتساق (Inconsistency) وانعدام القيود (Lack of Constraints)

Problem Two: Inconsistency and Lack of Constraints.

في نظام الملفات، كل ملف هو عالم منفصل بذاته. لا توجد آلية آلية تضمن أن sample_id المذكور في ملف expression_data.csv موجود فعلاً في ملف sample_metadata.csv. لا توجد طريقة لمنع إدخال قيمة غير منطقية في عمود age (مثل عمر سالب أو 200 سنة) بشكل موحد عبر جميع الملفات. لا توجد قاعدة

تمنع تكرار معرف عينة في نفس الملف أو عبر ملفات مختلفة.

In a file system, each file is a world unto itself. There is no automatic mechanism ensuring that the sample_id mentioned in expression_data.csv actually exists in sample_metadata.csv. There is no way to uniformly prevent entering an illogical value in the age column (like a negative age or 200 years) across all files. There is no rule preventing duplication of a sample ID within the same file or across different files.

هذا النقص في القيود (Constraints) هو مصدر دائم للأخطاء الصامتة. قد يكون البرنامج يعمل وينتج نتائج، لكنها مبنية على بيانات غير متسقة. تكتب سكربتاً يقرأ العينات من ملف، والقراءات من ملف آخر، ويفترض أن الترتيب متطابق. لكن إذا أضيفت عينة جديدة في المنتصف، أو حُذفت أخرى، ينهار كل شيء دون سابق إنذار. تصحيح هذه الأخطاء يتطلب

جهداً يدوياً هائلاً، وغالباً ما يمر دون أن يُكتشف حتى مراحل متقدمة من التحليل.

This lack of constraints is a constant source of silent errors. The program might run and produce results, but they are built on inconsistent data. You write a script that reads samples from one file, and reads counts from another, assuming the order is identical. But if a new sample is added in the middle, or another is deleted, everything collapses without warning. Correcting these errors requires enormous manual effort, and often goes undetected until advanced stages of analysis.

في قاعدة البيانات العلائقية، القيود هي جزء من التصميم:

- قيود المفتاح الأساسي (PRIMARY KEY): تمنع تكرار المعرفات.
- قيود المفتاح الأجنبي (FOREIGN KEY): تضمن أن كل قيمة مرجعية موجودة فعلاً في الجدول الأصلي.

· قيود التحقق (CHECK): تمنع القيم غير المنطقية (مثل age >= 0 و age < 150).

· قيود التفرد (UNIQUE): تمنع تكرار قيم معينة (مثل معرف المريض الخارجي).

· قيود عدم الفراغ (NOT NULL): تضمن وجود قيم أساسية لا غنى عنها.

In a relational database, constraints are part of the design:

· PRIMARY KEY constraints: Prevent duplicate identifiers.

· FOREIGN KEY constraints: Ensure every referenced value actually exists in the source table.

· CHECK constraints: Prevent illogical values (e.g., age >= 0 and age < 150).

· UNIQUE constraints: Prevent duplication of specific values (e.g., external patient ID).

· NOT NULL constraints: Ensure essential values are always present.

هذه القيود يتم فرضها تلقائياً بواسطة محرك قاعدة البيانات، قبل أن تصل البيانات إلى طبقة التحليل. هذا يحول اكتشاف الأخطاء من مرحلة "بعد التحليل" إلى مرحلة "عند الإدخال"، مما يوفر ساعات من التصحيح ويحمي سلامة الاستنتاجات العلمية.

These constraints are enforced automatically by the database engine, before data reaches the analysis layer. This shifts error detection from the "post-analysis" phase to the "at-ingestion" phase, saving hours of debugging and protecting the integrity of scientific conclusions.

المشكلة الثالثة: العلاقات (Relationships) والاستعلامات متعددة الجداول (Multi-Table Queries)

Problem Three: Relationships and Multi-Table Queries.

البيانات الجينومية بطبيعتها شبكة معقدة من الكيانات المترابطة. العينة لها مريض، والتجربة لها عدة تشغيلات، والتشغيل ينتج قراءات، والقراءات تُحاذى على جينوم مرجعي، والجينوم يحتوي على جينات، والجينات لها تسلسلات، والتسلسلات لها طفرات، والطفرات مرتبطة بأمراض أو بمقاومة للأدوية. هذه ليست مجرد مجموعة من الجداول المنفصلة؛ إنها شبكة معرفية واحدة.

Genomic data is inherently a complex network of interconnected entities. A sample has a patient, an experiment has several runs, a run produces reads, reads are aligned to a reference genome, the genome contains genes, genes have sequences, sequences have mutations, and mutations are linked to diseases or drug resistance. This is not just a collection of separate tables; it is a single knowledge network.

في نظام الملفات، لاسترجاع "جميع الطفرات في جين BRCA1 من عينات مرضى سرطان الثدي الذين تتراوح أعمارهم بين 40 و 50 سنة"، ستحتاج إلى:

1. قراءة ملف المرضى وتصفية الأعمار.
2. قراءة ملف العينات وتصفية المرضى من الخطوة السابقة.
3. قراءة ملف التسلسلات وتصفية العينات من الخطوة السابقة وجينات BRCA1.
4. قراءة ملف الطفرات وتصفية التسلسلات من الخطوة السابقة.
5. دمج كل هذه النتائج في الذاكرة.

In a file system, to retrieve "all mutations in the BRCA1 gene from breast cancer patient samples aged between 40 and 50", you would need to:

1. Read the patients file and filter by age.
2. Read the samples file and filter by patients from step 1.
3. Read the sequences file and filter by samples from step 2 and BRCA1 genes.
4. Read the mutations file and filter by sequences from step 3.
5. Merge all these results in memory.

هذه العملية مرهقة برمجياً، بطيئة، وعرضة للخطأ. كل خطوة تتطلب قراءة ملف كامل (أو جزء كبير منه) في الذاكرة، ثم التصفية، ثم الانضمام. مع ملايين السجلات، تصبح هذه العملية مستهلكة للذاكرة والوقت بشكل غير مقبول.

This process is programmatically cumbersome, slow, and error-prone. Each step requires reading an entire file (or a large part of it) into memory, then filtering, then joining. With millions of records, this process becomes unacceptably memory- and time-consuming.

في قاعدة البيانات العلائقية، هذا السؤال العلمي يترجم مباشرة إلى استعلام SQL واحد:

In a relational database, this scientific question translates directly into a single SQL query:

SQL

```
SELECT m.*
FROM Mutations m
JOIN Sequences s ON m.sequence_id = s.sequence_id
JOIN Samples sm ON s.sample_id = sm.sample_id
JOIN Patients p ON sm.patient_id = p.patient_id
WHERE p.age BETWEEN 40 AND 50
  AND p.disease = 'breast_cancer'
  AND s.gene_name = 'BRCA1';
```

هذا الاستعلام يعبر عن المنطق البيولوجي مباشرة. محرك قاعدة البيانات يتولى مسؤولية كيفية تنفيذه بكفاءة (اختيار أفضل ترتيب للانضمام، استخدام الفهارس، قراءة البيانات الضرورية فقط). أنت لا تحتاج إلى كتابة حلقة للانضمام يدوياً.

This query expresses the biological logic directly. The database engine takes responsibility for how to execute it efficiently (choosing the best join order,

using indexes, reading only necessary data). You do not need to write a loop to manually join.

خلاصة الفصل الأول: الملفات وسيلة ممتازة للتبادل والتخزين المؤقت، لكنها نظام إدارة بيانات فاشل. الفجوات الثلاث (التكرار، الاتساق، العلاقات) تجعل الملفات غير مناسبة للمشاريع التي تتجاوز بضع عشرات من السجلات. قاعدة البيانات العلائقية توفر حلاً منهجياً لهذه الفجوات، لكنها تتطلب نقلة في التفكير: من "قراءة ملف" إلى "تصميم مخطط" (Schema). هذه النقلة هي محور الأبواب القادمة.

Conclusion of Chapter One: Files are an excellent means of exchange and temporary storage, but they are a failed data management system. The three gaps (redundancy, consistency, relationships) make files unsuitable for projects exceeding a few dozen records. The relational database provides a systematic solution to these gaps, but it requires a shift in thinking: from

"reading a file" to "designing a Schema". This shift is the focus of the coming units.

الفصل الثاني: البيانات المهيكلة وشبه المهيكلة — من المرونة إلى الصرامة

Chapter Two: Structured and Semi-Structured Data — From Flexibility to Rigor

الشكل 2 — الخريطة المفاهيمية للفصل

Figure 2 — Conceptual map of the chapter

2.5 من القاموس إلى Schema — متى تصبح البنية المؤقتة بنية دائمة؟

الشكل 2 — سير العمل الهندسي للفصل

Figure 2 — Engineering workflow of the chapter

من الأدوات التي اكتسبتها في الجزء الثاني، استخدمت القواميس لتمثيل metadata مرنة. في الجزء الثالث، لا تتخلص من هذه الفكرة؛ بل تسأل أي أجزاء من القاموس تحتاج إلى هوية مستقلة وعلاقات وقيود. بعض البيانات ستظل JSON أو نصًا مرنًا، وبعضها يستحق أن يصبح أعمدة وجداول.

2.5 From Dictionaries to a Schema — When Does a Temporary Structure Become Persistent?

The knowledge built in Part Two leads directly here because you used dictionaries to represent flexible metadata. In Part Three, you do not discard that idea; instead, you ask which parts of the dictionary need independent identities, relationships, and constraints. Some data may remain JSON or flexible text, while other data deserves explicit columns and tables.

Python

```python
metadata = {
"instrument": "sequencer-A",
"run_id": "RUN-1042",
"read_length": 150,
"paired_end": True,
}
```

SQL

```sql
CREATE TABLE sequencing_runs (
    run_id TEXT PRIMARY KEY,
    instrument TEXT NOT NULL,
    read_length INTEGER NOT NULL,
    paired_end INTEGER NOT NULL CHECK (paired_end IN (0, 1))
);
```

لفهم لماذا تختلف قاعدة البيانات عن الملفات، يجب أن نصنف البيانات نفسها. في علم البيانات، نميز بين ثلاثة أنماط رئيسية، لكننا سنركز على النمطين الأكثر صلة

بالمعلوماتية الحيوية: شبه المهيكلة (Semi-Structured) والمهيكلة (Structured). النمط الثالث، غير المهيكل (Unstructured)، مثل الصور والنصوص الحرة، يتطلب معالجة مختلفة (مثل استخراج الملامح)، لكننا سنتجاوزه هنا لأنه ليس محور تركيزنا في هذا الباب.

To understand why a database differs from files, we must classify the data itself. In data science, we distinguish between three main patterns, but we will focus on the two most relevant to bioinformatics: Semi-Structured and Structured. The third pattern, Unstructured, such as images and free text, requires different processing (e.g., feature extraction), but we will skip it here as it is not our focus in this unit.

أولاً: البيانات شبه المهيكلة — مرونة JSON و GenBank

First: Semi-Structured Data — The Flexibility of JSON and GenBank.

البيانات شبه المهيكلة هي التي تحتوي على عناصر تنظيمية (مثل العلامات، الأقواس، الفواصل، الكلمات المفتاحية) لكنها لا تلتزم بنموذج صارم وثابت. كل كيان قد يحتوي على مجموعة مختلفة من الحقول. المثال الأكثر وضوحاً هو JSON (JavaScript Object Notation)، الذي أصبح المعيار الفعلي لتبادل البيانات بين واجهات برمجة التطبيقات (APIs).

Semi-structured data contains organizational elements (such as tags, brackets, delimiters, keywords) but does not adhere to a strict, fixed model. Each entity may contain a different set of fields. The most obvious example is JSON (JavaScript Object Notation), which has become the de facto standard for data exchange between APIs.

لنأخذ مثالاً من بيانات RNA-seq كما قد تظهر في استجابة API:

Let's take an example from RNA-seq data as it might appear in an API response:

JSON

```json
{
"sample_id": "S001",
"patient": {
"id": "P001",
"age": 45,
"sex": "F",
"diagnosis": "breast_cancer"
},
"experiment": {
"type": "RNA-seq",
"platform": "Illumina NovaSeq",
"run_date": "2025-03-15"
},
"counts": {
"BRCA1": 1245,
"TP53": 893,
"EGFR": 2104
}
}
```

لاحظ الخصائص المميزة للبيانات شبه المهيكلة هنا:

1. التداخل (Nesting): كائن patient يحتوي على كائنات فرعية. هذا يعكس البنية المنطقية للبيانات، لكنه يجعل الاستعلام صعباً في النماذج العلائقية.
2. المرونة (Flexibility): يمكن لعينة أخرى أن تحتوي على حقل treatment (العلاج) بينما هذه لا تحتويه. لا يوجد قيد إلزامي.
3. الحقول الاختيارية: counts قد تحتوي على جينات مختلفة في كل عينة حسب ما تم قياسه.
4. قابلية القراءة البشرية: الهيكل مفهوم للبشر، لكنه يتطلب تحليلاً (Parsing) برمجياً لاستخراج القيم.

Notice the distinguishing characteristics of semi-structured data here:

1. Nesting: The patient object contains sub-objects. This reflects the logical structure of the data, but makes querying difficult in relational models.
2. Flexibility: Another sample could contain a treatment field while this one does not. There is no mandatory constraint.

3. Optional fields: counts may contain different genes in each sample depending on what was measured.
4. Human readability: The structure is understandable to humans, but requires programmatic parsing to extract values.

بعد أن أصبحت الأساسيات السابقة مستقرة، تعاملت مع هذا النمط باستمرار. عندما كتبت:

Python

```
record.annotations = {"organism": "Homo sapiens", "tissue": "liver"}
```

فأنت كنت تنشئ بنية شبه مهيكلة. annotations هو قاموس (Dictionary) في Python، وهو مماثل لكائن JSON. يمكنك إضافة أي مفتاح تريده، وقراءة أي مفتاح موجود، وتجاهل المفقود. هذه المرونة هي قوة Biopython، لأن البيانات الوصفية للجينات تختلف اختلافاً كبيراً بين المصادر المختلفة.

Among the tools you gained in Part Two, you dealt with this pattern constantly. When you wrote:

Python

```
record.annotations = {"organism": "Homo sapiens", "tissue": "liver"}
```

you were creating a semi-structured structure. annotations is a Dictionary in Python, equivalent to a JSON object. You can add any key you want, read any existing key, and ignore missing ones. This flexibility is Biopython's strength, because gene metadata varies greatly between different sources.

لكن هذه المرونة لها ثمن هندسي: عند محاولة الإجابة عن سؤال مثل "ما هو متوسط عمر المرضى الذين لديهم طفرة في جين BRCA1؟"، ستحتاج إلى:

1. اجتياز جميع ملفات JSON أو جميع كائنات SeqRecord.
2. التحقق من وجود حقل annotations['patient_age'] في كل كائن.
3. إذا كان موجوداً، استخراجه وتخزينه مؤقتاً.
4. جمع القيم وحساب المتوسط.

5. إذا تغير هيكل JSON في المستقبل (مثلاً، أصبح age داخل patient بدلاً من أن يكون حقلاً مباشراً)، فسيتعين عليك تعديل الكود البرمجي لكل استعلام من هذا النوع.

However, this flexibility has an engineering cost: When trying to answer a question like "What is the average age of patients who have a mutation in the BRCA1 gene?", you would need to:

1. Traverse all JSON files or all SeqRecord objects.
2. Check for the existence of the annotations['patient_age'] field in each object.
3. If present, extract it and store it temporarily.
4. Collect the values and calculate the average.
5. If the JSON structure changes in the future (e.g., age moves inside patient instead of being a direct field), you will have to modify the programmatic code for every query of this type.

في مشروع صغير به 100 تسلسل، هذا مقبول. في نظام به مليون تسلسل، هذا غير قابل للتطبيق.

In a small project with 100 sequences, this is acceptable. In a system with a million sequences, this is infeasible.

ثانياً: البيانات المهيكلة — صرامة الجداول

Second: Structured Data — The Rigor of Tables.

البيانات المهيكلة، على النقيض، تتبع نموذجاً صارماً: جداول (Tables) ذات صفوف (Rows) و أعمدة (Columns). كل عمود له نوع بيانات (Data Type) محدد: DATE ،REAL ،TEXT ،INTEGER، إلخ. كل صف يجب أن يلتزم بهذا النموذج. الحقول الاختيارية يتم تمثيلها بـ NULL، لكن حتى NULL له معنى محدد (غياب القيمة) ويتم التعامل معه بشكل موحد.

Structured data, in contrast, follows a strict model: Tables with Rows and Columns. Each column has a specific Data Type: INTEGER, TEXT, REAL, DATE,

etc. Each row must conform to this model. Optional fields are represented by NULL, but even NULL has a specific meaning (absence of value) and is handled uniformly.

نفس البيانات السابقة، لكن في نموذج علاقي (جدولين):

The same data as before, but in a relational model (two tables):

جدول patients:

patient_id age sex diagnosis.

P001 45 F breast_cancer

جدول samples:

sample_id patient_id experiment_type platform run_date.

S001 P001 RNA-seq NovaSeq 2025-03-15

جدول counts:

sample_id gene count

S001 BRCA1 1245

S001 TP53 893

S001 EGFR 2104

Table patients:

patient_id age sex diagnosis.

P001 45 F breast_cancer

Table samples:

sample_id patient_id experiment_type platform run_date.

S001 P001 RNA-seq NovaSeq 2025-03-15

Table counts:

sample_id gene count

S001 BRCA1 1245

S001 TP53 893

S001 EGFR 2104

لاحظ الفروق الجوهرية:

1. التطبيع (Normalization): قمنا بتفكيك الكائن المتداخل إلى ثلاثة جداول مترابطة. كل حقيقة مخزنة في مكان واحد.
2. الصرامة (Rigor): age يجب أن يكون عدداً صحيحاً. diagnosis يجب أن يكون نصاً. لا يمكن إدخال عمر نصي أو تشخيص رقمي.
3. الاستعلام الموحد: السؤال عن متوسط عمر المرضى المصابين بـ BRCA1 يصبح استعلاماً واحداً (كما رأينا في الفصل الأول).
4. فصل المسؤوليات: جدول patients لا يعرف شيئاً عن counts، وجدول counts لا يعرف شيئاً عن patients، لكن الربط يتم عبر المفاتيح عند الحاجة.

Notice the fundamental differences:

1. Normalization: We decomposed the nested object into three interrelated tables. Each fact is stored in one place.
2. Rigor: age must be an integer. diagnosis must be text. You cannot insert a textual age or a numeric diagnosis.
3. Unified Querying: The question about the average age of BRCA1 patients becomes a single query (as we saw in Chapter One).
4. Separation of Concerns: The patients table knows nothing about counts, and the counts table knows nothing about patients, but linking is done via keys when needed.

أين يقع GenBank في هذا التصنيف؟

Where does GenBank fit in this classification?

GenBank هو حالة خاصة من البيانات شبه المهيكلة. ملف GenBank له بنية سطرية تعتمد على الكلمات المفتاحية (LOCUS، DEFINITION، ACCESSION، FEATURES، ORIGIN). هذه الكلمات

المفتاحية تعمل كعلامات (Tags) تحدد بداية ونهاية الأقسام. لكن داخل قسم FEATURES، هناك بنية متداخلة أخرى (ميزات داخل ميزات، مؤهلات ذات أسماء متغيرة). هذا يجعل GenBank مثالاً كلاسيكياً على البيانات شبه المهيكلة: لديه هيكل عام ثابت نسبياً، لكن داخله توجد مرونة كبيرة في المؤهلات والميزات. هذا هو السبب الذي جعل SeqFeature.qualifiers قاموساً في Biopython، وليس كائناً بصفات ثابتة.

GenBank is a special case of semi-structured data. A GenBank file has a line-based structure relying on keywords (LOCUS, DEFINITION, ACCESSION, FEATURES, ORIGIN). These keywords act as Tags defining the start and end of sections. However, inside the FEATURES section, there is another nested structure (features inside features, qualifiers with variable names). This makes GenBank a classic example of semi-structured data: it has a relatively fixed overall structure, but great internal flexibility in qualifiers and features. This is why SeqFeature.qualifiers is a dictionary in Biopython, not an object with fixed attributes.

الخلاصة الهندسية للفصل الثاني:

Engineering Conclusion of Chapter Two:

· استخدم البيانات شبه المهيكلة (JSON، قاموس Python، GenBank) عندما تحتاج إلى:

· مرونة عالية في الحقول.

· تغييرات متكررة في الهيكل.

· تبادل البيانات بين أنظمة مختلفة.

· تخزين مؤقت أو تكوينات (Configurations).

· استخدم البيانات المهيكلة (جداول SQL) عندما تحتاج إلى:

· استعلامات معقدة ومتكررة.

· سلامة بيانات صارمة (Constraints).

· أداء عالٍ في عمليات القراءة والتصفية.

· علاقات واضحة بين الكيانات.

· Use Semi-Structured data (JSON, Python dictionaries, GenBank) when you need:

· High flexibility in fields.

· Frequent structural changes.

· Data exchange between different systems.

· Temporary storage or configurations.

· Use Structured data (SQL tables) when you need:

· Complex and frequent queries.

· Strict data integrity (Constraints).

· High performance in read and filter operations.

· تاريخ طويل من البيانات (Provenance) يتطلب تتبعاً دقيقاً.

· Clear relationships between entities.

· Long data history (Provenance) requiring precise tracking.

الفصل الثالث: من الكائن إلى السجل — جسر OOP إلى SQL

Chapter Three: From Object to Record — The OOP to SQL Bridge

ليس تحويل أسماء الحقول فقط. عليك أن تسأل ما الذي يمثله الكائن، وما الذي يجب SQLو OOP الجسر بين
أن يبقى ثابتًا عبر الزمن، وما الذي يمثل علاقة بين كائنين، وما الذي يجب أن يخضع لقيد في قاعدة البيانات.
الكائن قد يحتوي سلوكًا وحالة مؤقتة، بينما السجل يمثل حالة قابلة للحفظ والاسترجاع. لذلك يصبح
Repository حدًا معماريًا مهمًا: فهو يمنع طبقة التحليل من معرفة تفاصيل SQL، ويجعل اختبار منطق
المجال ممكنًا حتى عند تغيير محرك التخزين لاحقًا.

The OOP-to-SQL bridge is not merely a conversion of field names. You must ask what the object represents, what must remain stable over time, what represents a relationship between two objects, and what should be enforced as a database constraint. An object may contain behavior and transient state, whereas a row represents a persistent state that can be stored and retrieved. The Repository therefore becomes an important architectural boundary: it keeps analytical code independent of SQL details and makes domain logic testable even if the storage engine changes later.

مختبر: تخزين كائن المجال مع إدارة الإصدارات

Laboratory: Persisting a Domain Object with Versioned Storage

يجب أن يبقى كائن المجال موجودًا بعد انتهاء عملية Python دون أن يفقد الهوية التي جعلته ذا معنى علمي. لذلك تخزن طبقة الاستمرارية المعرفات الثابتة والإصدارات والمصدر بصورة منفصلة عن الدوال المؤقتة.

A domain object should survive beyond one Python process without losing the identity that made it scientifically meaningful. The persistence layer therefore stores stable identifiers, versions, and provenance separately from transient methods.

يحتاج النظام الواقعي أيضًا إلى قاعدة تحديث حتمية. استبدال التسلسل الحالي لا يمحو السجل السابق؛ بل ينشئ إصدارًا جديدًا يمكن أن يختاره تشغيل التحليل صراحةً.

A real system also needs a deterministic update rule. Replacing the current sequence does not erase the earlier record; it creates a new version that can be selected explicitly by the analysis run.

الاستمرارية ذات الإصدارات داخل OOP

Versioned persistence in OOP

```
from dataclasses import dataclass
import sqlite3

@dataclass(frozen=True)
class SequenceSnapshot:
    accession: str
    version: int
    sequence: str

class SequenceRepository:
    def __init__(self, db_path: str):
        self.db_path = db_path

    def save(self, item: SequenceSnapshot) -> None:
        with sqlite3.connect(self.db_path) as con:
            con.execute("""
            CREATE TABLE IF NOT EXISTS sequence_versions (
                accession TEXT NOT NULL,
                version INTEGER NOT NULL,
                sequence TEXT NOT NULL,
                PRIMARY KEY (accession, version)
            )
            """)
            con.execute(
                "INSERT INTO sequence_versions VALUES (?, ?, ?)",
                (item.accession, item.version, item.sequence),
            )
```

الشكل 3 ــــ الخريطة المفاهيمية للفصل

Figure 3 — Conceptual map of the chapter

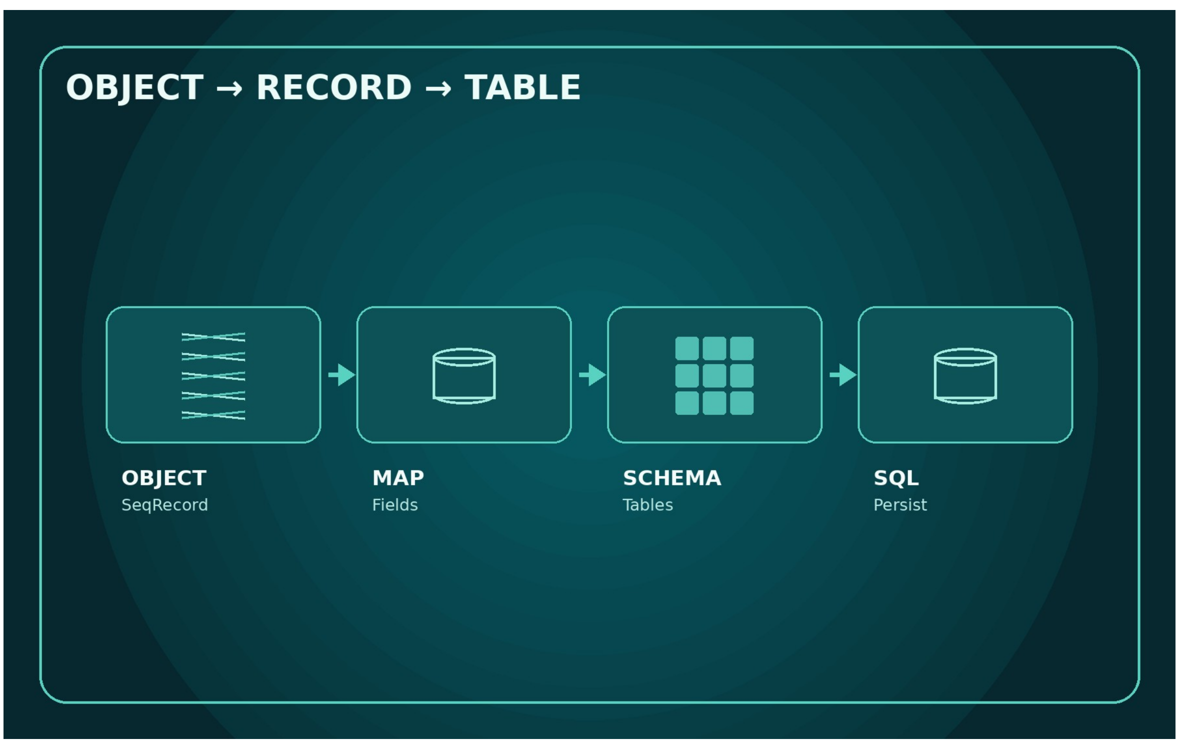
الشكل 3 ــــ سير العمل الهندسي للفصل

Figure 3 — Engineering workflow of the chapter

ما تعلّمته سابقًا عن البيانات والبرمجة يصبح هنا نقطة انطلاق، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

Once the earlier foundations are stable, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

3.5 SeqRecord to Three Relational Layers — The Complete Application.



3.5 SeqRecord إلى ثلاث طبقات علائقية ــــ التطبيق الكامل

Figure — OBJECT → RECORD → TABLE

خذ الآن كائن SeqRecord غنيًا بالهوية والتسلسل والميزات. لا تحفظه ككتلة غير قابلة للاستعلام. افصل الكيان الأساسي عن الميزات وعن المؤهلات التي تحتاج إلى البحث أو التحديث المستقل. هذا هو الجسر الحقيقي بين OOP والتصميم العلاقي.

Take a richly annotated SeqRecord containing identity, sequence, and features. Do not store it as an opaque block. Separate the core entity from features and qualifiers that need independent querying or updating. This is the real bridge between OOP and relational design.

Python

```
from dataclasses import dataclass, field


@dataclass
class Feature:
    start: int
    end: int
    feature_type: str
    qualifiers: dict[str, str] = field(default_factory=dict)


@dataclass
class SequenceRecord:
```

```
    accession: str
    sequence: str
    description: str
    features: list[Feature] = field(default_factory=list)
```

SQL

```sql
CREATE TABLE sequence_records (
    sequence_id INTEGER PRIMARY KEY,
    accession TEXT NOT NULL UNIQUE,
    sequence TEXT NOT NULL,
    description TEXT
);

CREATE TABLE sequence_features (
    feature_id INTEGER PRIMARY KEY,
    sequence_id INTEGER NOT NULL,
    start_pos INTEGER NOT NULL,
    end_pos INTEGER NOT NULL,
    feature_type TEXT NOT NULL,
    FOREIGN KEY (sequence_id)
        REFERENCES sequence_records(sequence_id)
);

CREATE TABLE feature_qualifiers (
    qualifier_id INTEGER PRIMARY KEY,
    feature_id INTEGER NOT NULL,
    qualifier_key TEXT NOT NULL,
    qualifier_value TEXT,
    FOREIGN KEY (feature_id)
        REFERENCES sequence_features(feature_id)
);
```

Python

```
def flatten_record(record: SequenceRecord):
    sequence_row = (
        record.accession,
        record.sequence,
        record.description,
    )

    feature_rows = [
        (feature.start, feature.end, feature.feature_type)
        for feature in record.features
    ]

    return sequence_row, feature_rows
```

لا يعني هذا أن كل SeqRecord يجب أن يتحول دائمًا إلى ثلاث جداول؛ التصميم يعتمد على أسئلة المشروع. المقصود هو أن تتعلم كيف تفكك الكائن إلى كيانات ذات معنى عندما يصبح الاستعلام المستقل ضرورة.

This does not mean that every SeqRecord must always become three tables; the design depends on the project's questions. The point is to learn how to decompose an object into meaningful entities when independent querying becomes necessary.

الخطوة التالية في هذا المسار تبدأ مما أُنجز في الجزء الثاني، أتقنت بناء كائنات Python باستخدام البرمجة الكائنية (OOP). كانت هذه الكائنات تعكس الكيانات البيولوجية: Seq للتسلسل، SeqRecord للتسلسل مع هويته وبياناته الوصفية، SeqFeature للموقع والاتجاه والميزة. كانت هذه الكائنات متجاورة (Cohesive): البيانات (Attributes) والسلوك (Methods) معاً في وحدة واحدة. كانت أيضاً مترابطة (Interconnected): SeqRecord يحتوي على Seq، و features قائمة تحتوي على SeqFeature.

What you learned earlier about data and programming now becomes the starting point, you mastered building Python objects using Object-Oriented Programming (OOP). These objects reflected biological entities: Seq for sequence, SeqRecord for sequence with identity and metadata, SeqFeature for location, strand, and feature. These objects were Cohesive: data (Attributes) and behavior (Methods) together in one unit. They were also Interconnected: SeqRecord contains Seq, and features is a list containing SeqFeature.

الآن، نحتاج إلى تخزين هذه الكائنات بشكل دائم. لكن قاعدة البيانات العلائقية لا تفهم التداخل (Nesting) ولا السلوك (Methods). هي تفهم الجداول المسطحة (Flat Tables) و العلاقات (Relationships). الانتقال من كائن متداخل إلى جداول

مسطحة يسمى التطبيع (Normalization). هذا ليس مجرد تحويل تقني؛ إنه إعادة تصميم (Redesign) لكيفية تمثيل البنية المنطقية للبيانات.

Now, we need to store these objects persistently. However, a relational database does not understand Nesting or Methods. It understands Flat Tables and Relationships. The transition from a nested object to flat tables is called Normalization. This is not just a technical conversion; it is a Redesign of how the logical structure of data is represented.

دعنا نأخذ مثالاً متكاملاً من الجزء الثاني. افترض أن لدينا هذا الكائن (مبسط قليلاً للتوضيح):

Let's take an integrated example from Part Two. Suppose we have this object (simplified slightly for clarity):

Python

```
from dataclasses import dataclass, field
from typing import List, Dict, Optional
```

```
@dataclass
class Feature:
    start: int
    end: int
    strand: int  # +1 for forward, -1 for reverse
    feature_type: str  # e.g., "exon", "CDS", "promoter"
    qualifiers: Dict[str, str]  # e.g., {"gene": "BRCA1", "note": "example"}

@dataclass
class Gene:
    gene_id: str
    sequence: str  # DNA string
    organism: str
    chromosome: str
    features: List[Feature] = field(default_factory=list)
```

هذا هو نموذج OOP المثالي. Gene يحتوي على sequence وorganism وchromosome وقائمة من features. كل Feature يحتوي على إحداثيات واتجاه ونوع ومؤهلات (Qualifiers) وهي قاموس مرن.

This is the ideal OOP model. Gene contains sequence, organism, chromosome, and a list of features. Each Feature contains coordinates, strand, type, and qualifiers (a flexible dictionary).

كيف نمثل هذا في SQLite (بنية علائقية)؟

How do we represent this in SQLite (relational structure)?

نحتاج إلى ثلاث جداول على الأقل، لأن لدينا ثلاثة مستويات منطقية: الجين (الأب)، والميزات (الأبناء)، والمؤهلات (أحفاد اختياريون). في أبسط صورها، سنستخدم جدولين (جين، وميزات)، وسنخزن المؤهلات كـ JSON نصي في عمود واحد، لأنها متغيرة البنية.

We need at least three tables, because we have three logical levels: the gene (parent), features (children), and qualifiers (optional grandchildren). In its simplest form, we will use two tables (genes and features), and store qualifiers as textual JSON in one column, as they are variable in structure.

SQL

```
-- الجدول الأول: الجينات (الجدول الأب)
CREATE TABLE genes (
    gene_id TEXT PRIMARY KEY,
    sequence TEXT NOT NULL,
    organism TEXT NOT NULL,
    chromosome TEXT NOT NULL
);

-- الجدول الثاني: الميزات (الجدول الابن)
CREATE TABLE features (
    feature_id INTEGER PRIMARY KEY AUTOINCREMENT,
    gene_id TEXT NOT NULL,
    start INTEGER NOT NULL,
    end INTEGER NOT NULL,
    strand INTEGER NOT NULL CHECK(strand IN (-1, 1)),
    feature_type TEXT NOT NULL,
    qualifiers TEXT,  -- JSON string, e.g., '{"gene": "BRCA1"}'
    FOREIGN KEY (gene_id) REFERENCES genes(gene_id) ON DELETE CASCADE
);
```

لاحظ الأمور التالية في هذا التصميم:

1. المفتاح الأساسي لـ genes: gene_id هو مفتاح طبيعي (Natural Key) لأنه يمثل المعرف الفريد في العالم الحقيقي. في بعض

التصاميم نفضل مفتاحاً صناعياً (Surrogate Key) مثل id INTEGER، لكننا هنا نستخدم المعرف الحقيقي للتبسيط ولأنه يتوافق مع استخدام SeqRecord.id.

2. المفتاح الأساسي لـ features: نستخدم feature_id كرقم متزايد (AUTOINCREMENT) لأنه لا يوجد معرف طبيعي فريد للميزات. نضيف gene_id كمفتاح أجنبي.

3. الربط: FOREIGN KEY تضمن أن كل ميزة تنتمي إلى جين موجود. ON DELETE CASCADE تعني أن حذف الجين سيحذف جميع ميزاته تلقائياً (مفيد لتنظيف البيانات).

4. التعامل مع المؤهلات: qualifiers هو عمود TEXT، لكنه يحتوي على سلسلة JSON. هذا حل وسط عملي: نحتفظ بمرونة Biopython (القاموس) لكن نخزنها كنص. عند الاسترجاع،

نحولها مرة أخرى إلى قاموس. هذا يسمح بإضافة مؤهلات جديدة دون تغيير هيكل الجدول.

Notice the following in this design:

1. Primary Key of genes: gene_id is a Natural Key because it represents the unique identifier in the real world. In some designs, we prefer a Surrogate Key such as id INTEGER, but we use the real identifier here for simplicity and because it aligns with SeqRecord.id.
2. Primary Key of features: We use feature_id as an auto-incrementing number because there is no natural unique identifier for features. We add gene_id as a foreign key.
3. Linking: FOREIGN KEY ensures that every feature belongs to an existing gene. ON DELETE CASCADE means deleting a gene will automatically delete all its features (useful for data cleanup).
4. Handling qualifiers: qualifiers is a TEXT column, but it contains a JSON string. This is a practical compromise: we keep the flexibility of Biopython (the dictionary) but store it as text. Upon retrieval, we convert it back to a dictionary. This allows adding new qualifiers without altering the table structure.

الآن، كيف نبني الجسر البرمجي بين OOP و SQL؟

Now, how do we build the programmatic bridge between OOP and SQL؟

الإجابة هي نمط المستودع (Repository Pattern). هذا النمط الهندسي يفصل منطق التطبيق (الذي يتعامل مع الكائنات) عن منطق التخزين (الذي يتعامل مع SQL). المستودع هو كائن OOP بحت، وظيفته الوحيدة هي ترجمة الكائنات إلى صفوف والعكس. كلما احتاج التطبيق إلى جين، يستدعي المستودع. كلما أراد حفظ جين، يمرر الكائن للمستودع.

The answer is the Repository Pattern. This engineering pattern separates application logic (which deals with objects) from storage logic (which deals with SQL). The Repository is a pure OOP object whose only function is to translate objects into rows and vice versa. Whenever the application needs a

gene, it calls the repository. Whenever it wants to save a gene, it passes the object to the repository.

لنكتب تطبيقاً واقعياً لهذا المستودع باستخدام sqlite3 المدمج في Python، مع التعامل مع JSON وحالات الخطأ بشكل واقعي. هذا الكود ليس مختصراً تعليمياً؛ إنه كود يمكن أن يعمل في نظام حقيقي (مع إضافة إدارة الاتصالات والـ Logging لاحقاً).

Let's write a realistic implementation of this repository using Python's built-in sqlite3, handling JSON and errors realistically. This code is not a shortened teaching snippet; it is code that could work in a real system (with connection management and logging added later).

Python

```
import sqlite3
import json
from typing import List, Optional, Dict
from dataclasses import dataclass, field


# تعريف الكائنات (OOP) كما في الجزء الثاني
# Define objects (OOP) as in Part Two
```

```
@dataclass
class Feature:
    start: int
    end: int
    strand: int
    feature_type: str
    qualifiers: Dict[str, str] = field(default_factory=dict)

@dataclass
class Gene:
    gene_id: str
    sequence: str
    organism: str
    chromosome: str
    features: List[Feature] = field(default_factory=list)

# تعريف المستودع (Repository) الذي يربط OOP بـ SQL
# Define the Repository that bridges OOP and SQL
class GeneRepository:
    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path
        self._initialize_schema()

    def _initialize_schema(self) -> None:
        """إنشاء الجداول إذا لم تكن موجودة.
        Create tables if they do not exist.
        """
        with sqlite3.connect(self.db_path) as conn:
            conn.execute("""
                CREATE TABLE IF NOT EXISTS genes (
                    gene_id TEXT PRIMARY KEY,
                    sequence TEXT NOT NULL,
```

```
            organism TEXT NOT NULL,
            chromosome TEXT NOT NULL
        )
    """)
    conn.execute("""
        CREATE TABLE IF NOT EXISTS features (
            feature_id INTEGER PRIMARY KEY AUTOINCREMENT,
            gene_id TEXT NOT NULL,
            start INTEGER NOT NULL,
            end INTEGER NOT NULL,
            strand INTEGER NOT NULL CHECK(strand IN (-1, 1)),
            feature_type TEXT NOT NULL,
            qualifiers TEXT,
            FOREIGN KEY (gene_id) REFERENCES genes(gene_id) ON DELETE CASCADE
        )
    """)
    # إنشاء فهرس لتسريع البحث عن الميزات حسب gene_id
    # Create an index to speed up feature lookup by gene_id
    conn.execute("CREATE INDEX IF NOT EXISTS idx_features_gene_id ON features(gene_id)")
    conn.commit()

def save(self, gene: Gene) -> None:
    """حفظ جين وميزاته في قاعدة البيانات (إدراج أو تحديث).
    Save a gene and its features to the database (insert or replace).
    """
    if not gene or not gene.gene_id:
        raise ValueError("Gene object must have a valid gene_id")

    with sqlite3.connect(self.db_path) as conn:
        # 1. حفظ الجين الرئيسي (مع REPLACE للتحديث)
        conn.execute(
            "INSERT OR REPLACE INTO genes (gene_id, sequence, organism, chromosome) VALUES
```

```
(?, ?, ?, ?)",
            (gene.gene_id, gene.sequence, gene.organism, gene.chromosome)
        )

        # 2. حذف الميزات القديمة المرتبطة بهذا الجين (لتجنب التكرار والفساد)
        conn.execute("DELETE FROM features WHERE gene_id = ?", (gene.gene_id,))

        # 3. حفظ الميزات الجديدة (إن وجدت)
        if gene.features:
            # تحضير البيانات للدفع الجماعي (Batch Insert) لكفاءة أفضل
            # Prepare data for batch insertion for better performance
            data = [
                (
                    gene.gene_id,
                    f.start,
                    f.end,
                    f.strand,
                    f.feature_type,
                    json.dumps(f.qualifiers) if f.qualifiers else None
                )
                for f in gene.features
            ]
            conn.executemany(
                "INSERT INTO features (gene_id, start, end, strand, feature_type, qualifiers) VALUES
(?, ?, ?, ?, ?, ?)",
                data
            )
        conn.commit()

    def find_by_id(self, gene_id: str) -> Optional[Gene]:
        """استرجاع جين كامل (مع ميزاته) من قاعدة البيانات بواسطة المعرف.
        Retrieve a complete gene (with its features) from the database by ID.
```

```
        """
        if not gene_id:
            return None

        with sqlite3.connect(self.db_path) as conn:
            conn.row_factory = sqlite3.Row  # يسمح بالوصول إلى الأعمدة بالاسم

            # استعلام الجين الأساسي
            gene_row = conn.execute(
                "SELECT gene_id, sequence, organism, chromosome FROM genes WHERE gene_id = ?",
                (gene_id,)
            ).fetchone()

            if not gene_row:
                return None

            # استعلام الميزات المرتبطة
            feature_rows = conn.execute(
                "SELECT start, end, strand, feature_type, qualifiers FROM features WHERE gene_id = ?
ORDER BY start",
                (gene_id,)
            ).fetchall()

            # إعادة بناء قائمة الميزات ككائنات
            features: List[Feature] = []
            for row in feature_rows:
                qualifiers = json.loads(row["qualifiers"]) if row["qualifiers"] else {}
                features.append(
                    Feature(
                        start=row["start"],
                        end=row["end"],
                        strand=row["strand"],
```

```
                feature_type=row["feature_type"],
                qualifiers=qualifiers
            )
        )

        return Gene(
            gene_id=gene_row["gene_id"],
            sequence=gene_row["sequence"],
            organism=gene_row["organism"],
            chromosome=gene_row["chromosome"],
            features=features
        )

    def delete(self, gene_id: str) -> None:
        """حذف جين وميزاته بالكامل (بسبب ON DELETE CASCADE).
        Delete a gene and all its features completely (due to ON DELETE CASCADE).
        """
        if not gene_id:
            return
        with sqlite3.connect(self.db_path) as conn:
            conn.execute("DELETE FROM genes WHERE gene_id = ?", (gene_id,))
            conn.commit()

    def find_by_organism(self, organism: str) -> List[Gene]:
        """استرجاع جميع الجينات لكائن حي معين (دون ميزاتها للتبسيط، أو بميزاتها حسب الحاجة).
        Retrieve all genes for a given organism (without features for simplicity, or with features if needed).
        """
        # هذا التطبيق يعيد الجينات بدون ميزات لتجنب جلب بيانات ضخمة. يمكن تعديله لاسترجاع الميزات عند الحاجة.
        # This implementation returns genes without features to avoid fetching massive data. Can be modified to fetch features if needed.
        with sqlite3.connect(self.db_path) as conn:
```

```
        conn.row_factory = sqlite3.Row
        rows = conn.execute(
            "SELECT gene_id, sequence, organism, chromosome FROM genes WHERE organism = ?",
            (organism,)
        ).fetchall()
        return [
            Gene(
                gene_id=r["gene_id"],
                sequence=r["sequence"],
                organism=r["organism"],
                chromosome=r["chromosome"],
                features=[]  # يمكن تطبيقه لاحقا (Lazy Loading) تم التحميل الكسول
            )
            for r in rows
        ]
```

تحليل هندسي لهذا الكود:

Engineering Analysis of this Code:

1. فصل المسؤوليات (Separation of Concerns): الكود لا يخلط بين منطق التطبيق ومنطق التخزين. Gene و Feature يعرفان كيف يمثلان أنفسهما منطقياً. GeneRepository يعرف كيف يخزن ويستعيد. أي تغيير في هيكل قاعدة البيانات (مثل إضافة

عمود) يتطلب تعديلاً في المستودع فقط، وليس في جميع أنحاء التطبيق.

2. التعامل مع المتغيرات (Handling Variability): استخدام json.dumps و json.loads للمؤهلات (Qualifiers) يحافظ على مرونة Biopython. يمكن إضافة أي مفتاح في qualifiers دون تعديل هيكل الجدول. هذا حل عملي يرضي كلاً من صرامة SQL ومرونة Python.

3. الكفاءة (Efficiency): استخدام executemany للإدراج الجماعي أسرع من حلقات execute الفردية. إنشاء فهرس idx_features_gene_id يسرع استعلامات الربط. التحقق من CHECK(strand IN (-1, 1)) يضمن صحة البيانات على مستوى قاعدة البيانات.

4. سلامة البيانات (Data Integrity): ON DELETE CASCADE يضمن عدم وجود ميزات يتيمة (Orphaned Features). INSERT OR REPLACE يسمح بالتحديث بسهولة دون الحاجة إلى كتابة استعلامات UPDATE منفصلة.

6. Separation of Concerns: The code does not mix application logic with storage logic. Gene and Feature know how to represent themselves logically. GeneRepository knows how to store and retrieve. Any change in the database structure (e.g., adding a column) requires modification only in the repository, not throughout the application.

7. Handling Variability: Using json.dumps and json.loads for qualifiers preserves Biopython's flexibility. Any key can be added to qualifiers without altering the table structure. This is a practical solution satisfying both SQL's rigor and Python's flexibility.

8. Efficiency: Using executemany for batch insertion is faster than individual execute loops. Creating the idx_features_gene_id index speeds up join queries. The CHECK(strand IN (-1, 1)) constraint ensures data correctness at the database level.

5. التحميل الكسول (Lazy Loading): في find_by_organism، لم نقم بتحميل الميزات لتجنب جلب كميات هائلة من البيانات إذا كان هناك العديد من الجينات. هذا نمط شائع: تحميل البيانات الأساسية فقط، وتحميل التفاصيل عند الطلب (يمكن إضافة معامل load_features للتحكم).

9. Data Integrity: ON DELETE CASCADE ensures no orphaned features exist. INSERT OR REPLACE allows easy updating without writing separate UPDATE queries.

10. Lazy Loading: In find_by_organism, we did not load features to avoid fetching massive data if there are many genes. This is a common pattern: load only base data, and load details on demand (a load_features parameter could be added for control).

الربط مع Biopython العملي:

Connecting to practical Biopython:

عندما نصل إلى هذه الطبقة الهندسية، كنت تقرأ ملفات GenBank وتحصل على SeqRecord. هذا الكائن يحتوي على seq (كائن Seq)، و features (قائمة من SeqFeature). يمكننا بسهولة كتابة محول (Adapter) يحول SeqRecord إلى Gene الخاص بنا، ثم يحفظه في قاعدة البيانات باستخدام GeneRepository.

The next engineering step grows out of the work completed in Part Two, you read GenBank files and obtained SeqRecord. This object contains seq (a Seq object) and features (a list of SeqFeature). We can easily write an Adapter that converts SeqRecord to our Gene, then saves it to the database using GeneRepository.

Python

```
from Bio import SeqIO
from Bio.SeqFeature import SeqFeature


def convert_seqrecord_to_gene(seq_record: SeqIO.SeqRecord) -> Gene:
    """تحويل SeqRecord من Biopython إلى كائن Gene الخاص بنا.
    Convert a Biopython SeqRecord to our Gene object.
    """
    # استخراج البيانات الوصفية الأساسية (قد تختلف حسب المصدر)
    annotations = seq_record.annotations
    organism = annotations.get("organism", "unknown")
    # قد يكون الكروموسوم مخزناً في مكان مختلف في GenBank،
    chromosome = annotations.get("chromosome", "unknown")

    features: List[Feature] = []
```

```
  for sf in seq_record.features:
      #  استخراج الإحداثيات (Biopython تستخدم 0-based، نحتفظ بها)
      location = sf.location
      start = location.start
      end = location.end
      strand = location.strand if location.strand else 1  # Biopython تستخدم None للشريط غير المعروف

      #  المؤهلات (Qualifiers) هي قاموس جاهز
      qualifiers = sf.qualifiers

      features.append(
          Feature(
              start=start,
              end=end,
              strand=strand,
              feature_type=sf.type,
              qualifiers=qualifiers
          )
      )

  return Gene(
      gene_id=seq_record.id,
      sequence=str(seq_record.seq),  # Seq إلى نص
      organism=organism,
      chromosome=chromosome,
      features=features
  )

#  مثال على الاستخدام الفعلي
#  Example of actual usage
#  record = SeqIO.read("example.gb", "genbank")
#  repo = GeneRepository("genomic_data.db")
```

```
# gene_obj = convert_seqrecord_to_gene(record)
# repo.save(gene_obj)
# retrieved = repo.find_by_id(gene_obj.gene_id)
# print(f"Retrieved gene with {len(retrieved.features)} features")
```

الخلاصة النهائية للباب الأول:

Final Conclusion of Unit One:

لقد حققنا ثلاث نقاط تحول في هذا الباب:

1. إدراك القيود: الملفات تفشل في إدارة العلاقات والتكرار والاتساق.
2. تصنيف البيانات: فهمنا الفرق بين شبه المهيكل (المرن) والمهيكل (الصارم)، وأدركنا متى نستخدم كل نمط.
3. بناء الجسر: أنشأنا طبقة مستودع (Repository) تحول كائنات OOP إلى سجلات SQL والعكس، باستخدام SQLite و JSON، مع الحفاظ على هوية الكائنات وعلاقاتها.

We have achieved three turning points in this unit:

1. Realizing Limitations: Files fail in managing relationships, redundancy, and consistency.
2. Classifying Data: We understood the difference between semi-structured (flexible) and structured (rigid), and realized when to use each pattern.
3. Building the Bridge: We created a Repository layer that converts OOP objects to SQL records and vice versa, using SQLite and JSON, while preserving object identity and relationships.

الانتقال إلى الباب الثاني:

Transition to Unit Two:

الآن وقد أصبحت لديك آلية ملموسة لحفظ الكائنات (Repository) وفهمت لماذا هي أفضل من الملفات، حان الوقت للتفكير في تصميم المخطط (Schema Design) نفسه. كيف نختار الكيانات المناسبة؟ كيف نحدد العلاقات بينها

(1:N، M:N)؟ كيف نختار المفاتيح الأساسية والأجنبية؟ هذه الأسئلة هي جوهر التفكير العلاقي (Relational Thinking)، وهي موضوع الباب الثاني. سننتقل من "كيف نخزن البيانات" إلى "ما هي البنية المنطقية للبيانات التي نحتاجها للإجابة عن أسئلتنا العلمية؟".

Now that you have a concrete mechanism for saving objects (Repository) and understand why it is better than files, it is time to think about Schema Design itself. How do we choose the right entities? How do we define relationships between them (1:N, M:N)? How do we choose primary and foreign keys? These questions are the essence of Relational Thinking, and they are the topic of Unit Two. We will move from "how to store data" to "what is the logical structure of data we need to answer our scientific questions?".

—

الباب الثاني: التفكير العلاقي — هندسة الكيانات والعلاقات والمخطط

Unit Two: Relational Thinking — Engineering Entities, Relationships, and Schema

الغرض من هذا الباب:

تحويل التفكير من "كيف نخزن البيانات" إلى "ما هي البنية المنطقية للبيانات". قبل كتابة أي أمر SELECT، يجب أن ترسم خريطة للكيانات التي تمثل عالمك البيولوجي، والعلاقات التي تربطها، والمفاتيح التي تحدد هويتها. هذا الباب هو الأساس الذي يمنع الفوضى في قواعد البيانات، ويجعل SQL أداة منطقية وليس مجموعة أوامر محفوظة.

Purpose of this unit:

To shift thinking from "how to store data" to "what is the logical structure of data." Before writing any SELECT statement, you must map the entities representing your biological world, the relationships connecting them, and the keys defining their identity. This unit is the foundation that prevents chaos in databases, making SQL a logical tool rather than a set of memorized commands.

الفصل الرابع: الكيان (**Entity**) — الجوهر المنطقي للبيانات

Chapter Four: The Entity — The Logical Essence of Data

الشكل 4 — الخريطة المفاهيمية للفصل

Figure 4 — Conceptual map of the chapter

4.5 من الوصف البيولوجي إلى DDL — أول جدول حقيقي

الشكل 4 — سير العمل الهندسي للفصل

Figure 4 — Engineering workflow of the chapter

بعد تحديد الكيان، تحتاج إلى ترجمة خصائصه إلى أعمدة وأنواع وقيود. لا تكتب الجدول باعتباره قائمة أسماء فقط؛ اسأل ما الذي يجب أن يكون فريدًا، وما الذي لا يجوز أن يكون فارغًا، وما الذي يعتمد على كيان آخر.

4.5 From Biological Description to DDL — The First Real Table.

After identifying an entity, translate its properties into columns, types, and constraints. Do not treat a table as a list of names; ask what must be unique, what cannot be missing, and what depends on another entity.

SQL

```
CREATE TABLE genes (
   gene_id INTEGER PRIMARY KEY,
   symbol TEXT NOT NULL UNIQUE,
   organism TEXT NOT NULL,
   chromosome TEXT,
   CHECK (length(symbol) > 0)
);
```

الخبرة السابقة في Python وBiopython تمهّد مباشرة لهذه الخطوة، صممت كائنات (Classes) تعكس العالم البيولوجي. صممت SeqRecord ليمثل تسلسلاً مع هويته، و SeqFeature ليمثل موقعاً وميزة، و Sample ليمثل عينة، و Experiment ليمثل تجربة تسلسل. هذه الكائنات كانت تمثل كيانات (Entities) في الذاكرة. في عالم قواعد البيانات، الكيان يتحول إلى جدول (Table). كل صف (Row) في الجدول هو مثيل (Instance) من ذلك الكيان، مثل كائن في قائمة Python.

When we reach this engineering layer, you designed Classes reflecting the biological world. You designed SeqRecord to represent a sequence with its identity, SeqFeature to represent a location and feature, Sample to represent a sample, and Experiment to represent a sequencing experiment. These objects represented Entities in memory. In the database world, an entity transforms

into a Table. Each Row in the table is an Instance of that entity, like an object in a Python list.

الكيان (Entity) هو شيء ملموس في العالم الحقيقي، يمكن تمييزه عن غيره، وله مجموعة من الصفات (Attributes) التي تصفه. الصفات تصبح أعمدة (Columns) في الجدول. اختيار الكيانات الصحيحة هو الخطوة الأكثر أهمية في تصميم قاعدة البيانات. إذا أخطأت في تعريف الكيانات، فإن كل استعلام لاحق سيكون معقداً أو مستحيلاً.

An Entity is a tangible thing in the real world, distinguishable from others, with a set of Attributes that describe it. Attributes become Columns in the table. Choosing the right entities is the most critical step in database design. If you misdefine entities, every subsequent query becomes complex or impossible.

دعنا نستعرض الكيانات الأساسية في أي مشروع معلوماتي حيوي، مستعينين بما تعلمناه من الجزء الثاني، ونضيف إليها كيانات إدارية (مثل المستخدمين) لأغراض الأمن والمراجعة:

Let's review the fundamental entities in any bioinformatics project, drawing from Part Two, and add administrative entities (like users) for security and auditing purposes:

Patient الفرد صاحب العينة. يحتوي على بيانات حساسة (يُفصل عن العينة لأسباب أخلاقية وقانونية). patient_id (PK), external_id (مجهول الهوية), age, sex, diagnosis. (جديد، لتمثيل السياق السريري)

الكيان (Entity) الوصف (Description) صفات نموذجية (Typical Attributes) المرجع في الجزء الثاني (Reference in Part Two)

Sample المادة البيولوجية المستخلصة (دم، نسيج، DNA/RNA).

sample_id (PK), patient_id (FK), tissue_type, collection_date, rna_quality (RIN).

يمثل مفهوم Sample المستخدم في تحليل RNA-seq.

Experiment مجموعة من الخطوات المنهجية المطبقة على عينة.

تحدد التقنية والبروتوكول. experiment_id (PK), sample_id (FK),

technology (RNA-seq, WGS, etc.), library_type, sequencing_platform. يمثل

سياق التجربة الذي يربط العينة بالقراءات.

Run تشغيل فعلي على جهاز التسلسل. ينتج ملف FASTQ واحداً

أو اثنين. run_id (PK), experiment_id (FK), run_date, instrument,

read_count, total_bases. يمثل الملفات الخام التي نتعامل معها

في SeqIO.parse().

Sequence تسلسل الحروف النيتروجينية (DNA/RNA) أو الأحماض

الأمينية (Protein). sequence_id (PK), run_id (FK) or gene_id (FK),

sequence_text (TEXT), length, gc_content. يمثل كائن Seq نفسه، لكن معزولاً كهوية قابلة للتخزين.

Gene وحدة وراثية ذات وظيفة محددة، تتكون من إكسونات وإنترونات. gene_id (PK), sequence_id (FK) (التسلسل المرجعي), symbol, description. يمثل كائن SeqRecord الذي يحمل معرف الجين (id).

Pathway مسار حيوي يشارك فيه عدة جينات. pathway_id (PK), name, description, source_db. (جديد، لتمثيل الشبكات البيولوجية M:N).

Feature موقع محدد على التسلسل (إكسون، CDS، موقع ارتباط). feature_id (PK), sequence_id (FK) or gene_id (FK), start, end, strand, type. يمثل SeqFeature مع إحداثياته واتجاهه.

User شخص يتفاعل مع النظام (لأغراض التوثيق والمراجعة). user_id (PK), username, role (admin, researcher, auditor). يتوافق مع مفهوم actor في سجلات التدقيق (Audit Logs).

AuditLog سجل زمني لكل عملية حساسة. audit_id (PK), user_id (FK), action, table_name, record_id, timestamp. يتوافق مع مفهوم audit_log الذي أنشأناه في biobrick_security.

لاحظ أن التسلسل و الجين و الميزة كيانات منفصلة. في SeqRecord، كانوا مجتمعين في كائن واحد. هذا الفصل هو جوهر التطبيع (Normalization): فصل الحقائق المستقلة لتجنب التكرار.

Notice that Sequence, Gene, and Feature are separate entities. In SeqRecord, they were combined in one object. This separation is the essence of Normalization: separating independent facts to avoid duplication.

تمثيل الكيانات في OOP (Python) قبل الانتقال إلى SQL:

Representing Entities in OOP (Python) before moving to SQL:

Python

```
from dataclasses import dataclass, field
from typing import Optional, List
from enum import Enum

# استخدام Enum للقيم الثابتة لضمان الصحة والقراءة
# Using Enum for fixed values to ensure correctness and readability
class Strand(Enum):
    FORWARD = 1
    REVERSE = -1

class Sex(Enum):
    MALE = "M"
    FEMALE = "F"
    OTHER = "O"

class SampleStatus(Enum):
    CONTROL = "control"
    TREATED = "treated"
    DISEASED = "diseased"

@dataclass
class Patient:
    patient_id: int  # معرف داخلي رقمي (Surrogate Key)
    external_id: str  # معرف خارجي لإخفاء الهوية
```

```
    age: Optional[int] = None
    sex: Optional[Sex] = None
    diagnosis: Optional[str] = None

@dataclass
class Sample:
    sample_id: int
    patient_id: int  # Foreign Key إلى Patient
    tissue_type: str
    collection_date: str
    rna_quality: Optional[float] = None  # RIN score
    status: SampleStatus = SampleStatus.CONTROL

@dataclass
class Feature:
    feature_id: int
    start: int
    end: int
    strand: Strand
    feature_type: str  # "exon", "CDS", "promoter"
    qualifiers: dict = field(default_factory=dict)  # JSON-like flexibility

@dataclass
class Gene:
    gene_id: int
    symbol: str
    description: Optional[str] = None
    # في OOP، نستخدم قائمة للعلاقة 1:N. في SQL، سنستخدم جدولاً منفصلا.
    features: List[Feature] = field(default_factory=list)
```

هذا الكود OOP يعكس البنية المنطقية. لاحظ أن Gene يحتوي على features كقائمة. هذا يعبر عن العلاقة "واحد إلى متعدد" (1:N) في لغة البرمجة. في قاعدة البيانات، سنعكس هذه العلاقة عبر مفتاح أجنبي في جدول features يشير إلى gene_id.

This OOP code reflects the logical structure. Note that Gene contains features as a list. This expresses the "One-to-Many" (1:N) relationship in programming language. In the database, we will reflect this relationship via a Foreign Key in the features table pointing to gene_id.

الفصل الخامس: العلاقات بين الكيانات — رسم خريطة الترابط

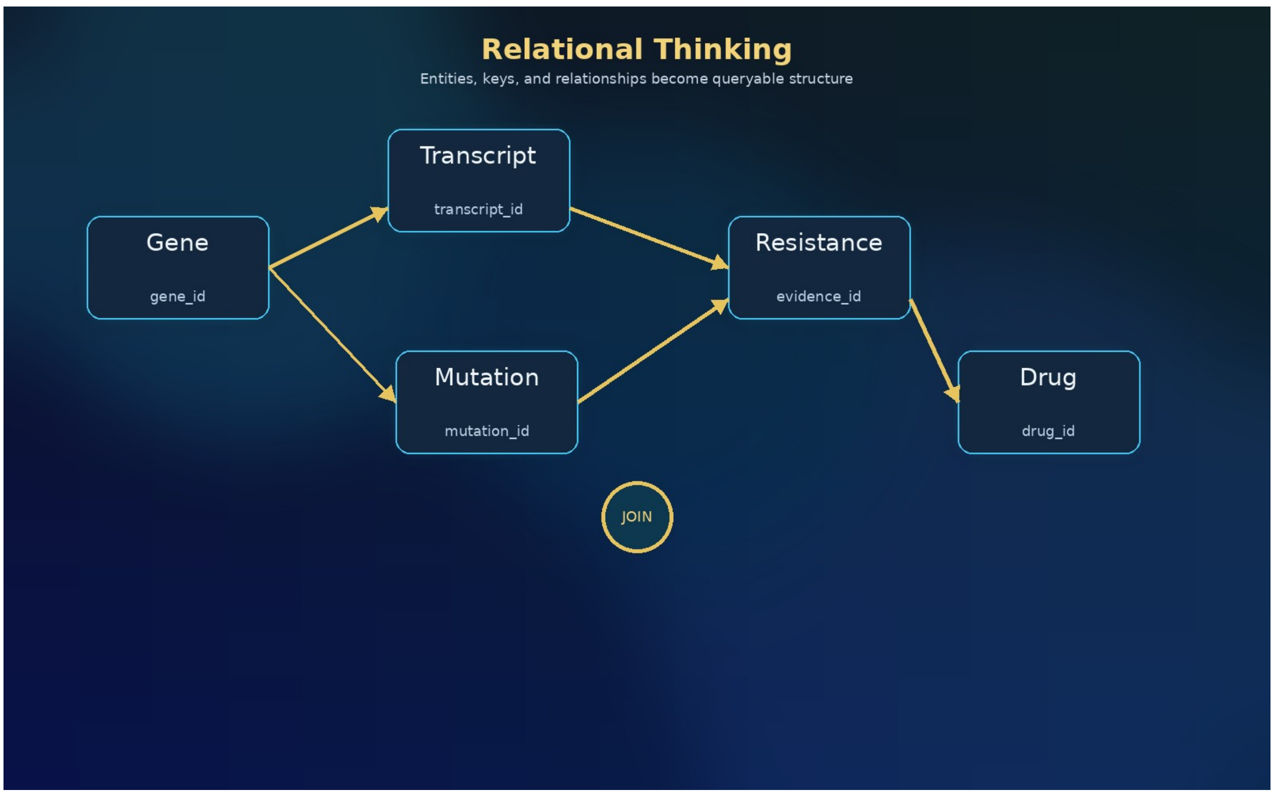
Chapter Five: Relationships Between Entities — Mapping Interconnection

الشكل 5 — الخريطة المفاهيمية للفصل

Figure 5 — Conceptual map of the chapter

الشكل 5 — سير العمل الهندسي للفصل

Figure 5 — Engineering workflow of the chapter



العلاقات بين الكيانات ليست تفاصيل ثانوية؛ إنها البنية التي تجعل السؤال البيولوجي قابلاً للاستعلام.

Relationships are not secondary details; they are the structure that makes biological questions queryable.

الانتقال إلى هذا المفهوم يبدأ من مهارة سابقة لديك، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

Your earlier experience with Python and Biopython prepares you directly for this step, you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

5.5 العلاقات في العالم البيولوجي — من الواقع إلى النموذج العلاقي

5.5 Relationships in the Biological World — From Reality to the Relational Model.

عندما تنظر إلى مشروع جينومي حقيقي، لا ترى صفوفًا منفصلة؛ ترى كيانات ترتبط ببعضها: عينة تنتمي إلى تجربة، والتجربة تنتج تشغيلات، والتشغيل يرتبط بملفات،

والتسلسل ينتمي إلى كائن أو مرجع، والطفرات قد ترتبط بتسلسلات وبنتائج مقاومة. قبل كتابة JOIN، تعلّم أن ترسم هذه العلاقات ذهنيًا.

When you look at a real genomic project, you do not see isolated rows; you see entities connected to one another: a sample belongs to an experiment, an experiment produces runs, a run is associated with files, a sequence belongs to an organism or reference, and mutations may connect sequences with resistance observations. Before writing JOIN, learn to reason about these relationships explicitly.

العلاقة one-to-many تعني أن سجلًا واحدًا في الطرف الأول يمكن أن يرتبط بعدة سجلات في الطرف الثاني. أما many-to-many فتحتاج عادةً إلى جدول وسيط. في البيانات البيولوجية، علاقة التجربة بالعينات مثال واضح على one-to-many، بينما علاقة الجينات بالمقاومات أو المسارات قد تحتاج إلى جدول ربط لأن كل طرف يمكن أن يرتبط بعدة عناصر من الطرف الآخر.

A one-to-many relationship means that one record on the first side can be associated with many records on the second side. A many-to-many relationship usually requires an associative table. In biological data, an experiment-to-samples relationship is a clear one-to-many example, while genes-to-resistance observations may require a linking table because each side can be associated with many elements of the other.

SQL

```sql
CREATE TABLE experiments (
    experiment_id INTEGER PRIMARY KEY,
    name TEXT NOT NULL
);

CREATE TABLE samples (
    sample_id INTEGER PRIMARY KEY,
    experiment_id INTEGER NOT NULL,
    sample_name TEXT NOT NULL,
    FOREIGN KEY (experiment_id)
        REFERENCES experiments(experiment_id)
);

CREATE TABLE gene_resistance (
    gene_id INTEGER NOT NULL,
    resistance_id INTEGER NOT NULL,
    PRIMARY KEY (gene_id, resistance_id)
);
```

لاحظ أن الجدول الوسيط لا يكرر بيانات الجين ولا بيانات المقاومة؛ إنه يخزن العلاقة نفسها. هذه الفكرة ستجعل JOIN في الباب التالي نتيجة طبيعية لتصميمك، لا مهارة نحوية منفصلة.

Notice that the associative table does not duplicate gene data or resistance data; it stores the relationship itself. This idea will make JOIN in the next unit a natural consequence of the design rather than an isolated piece of syntax.

الكيانات نادراً ما تعيش بمعزل عن بعضها. العينة تنتمي إلى مريض، والتجربة تطبق على عينة، والتشغيل ينتج عن تجربة، والتسلسل يُستخرج من تشغيل، والجين له تسلسل مرجعي وميزات متعددة، والجينات تتفاعل في مسارات حيوية. فهم هذه العلاقات هو جوهر النمذجة العلائقية. هناك ثلاثة أنواع أساسية من العلاقات.

Entities rarely exist in isolation. A sample belongs to a patient, an experiment is applied to a sample, a run results from an experiment, a sequence is extracted from a run, a gene has a reference sequence and multiple features, and genes interact in biological pathways. Understanding these relationships is the essence of relational modeling. There are three fundamental types of relationships.

النوع الأول: واحد إلى واحد (One-to-One — 1:1)

Type One: One-to-One (1:1)

في هذه العلاقة، كل سجل في الجدول الأول يرتبط بسجل واحد فقط في الجدول الثاني، والعكس صحيح. هذا نادر في البيانات الجينومية، لكنه يظهر في حالات مثل: "مريض واحد له سجل سريري واحد تفصيلي". لماذا نفصلها إذا كانت 1:1؟ لأسباب أمنية أو أدائية. السجل السريري قد يحتوي على

بيانات حساسة جداً (العنوان، رقم الهاتف) لا نريد أن تظهر في كل استعلام عن المريض. فصله في جدول منفصل يسمح بالتحكم في الوصول إليه بشكل أدق.

In this relationship, each record in the first table relates to exactly one record in the second, and vice versa. This is rare in genomic data, but appears in cases like: "one patient has one detailed clinical record." Why separate them if they are 1:1? For security or performance reasons. The clinical record may contain highly sensitive data (address, phone number) that we do not want to appear in every patient query. Separating it into its own table allows finer access control.

التنفيذ في SQL: نضع مفتاحاً أجنبياً في أحد الجدولين، مع جعله UNIQUE لضمان عدم تكرار الارتباط.

Implementation in SQL: Place a foreign key in one of the tables, making it UNIQUE to ensure the link is not duplicated.

SQL

```sql
CREATE TABLE patients (
    patient_id INTEGER PRIMARY KEY AUTOINCREMENT,
    external_id TEXT UNIQUE NOT NULL,
    age INTEGER,
    sex TEXT CHECK(sex IN ('M', 'F', 'O'))
);

-- الجدول المرتبط 1:1 (سجل سريري مفصل)
CREATE TABLE clinical_records (
    record_id INTEGER PRIMARY KEY AUTOINCREMENT,
    patient_id INTEGER UNIQUE NOT NULL, -- UNIQUE يضمن 1:1
    address TEXT,
    phone TEXT,
    emergency_contact TEXT,
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id) ON DELETE CASCADE
);
```

النوع الثاني: واحد إلى متعدد (One-to-Many — 1:N)

Type Two: One-to-Many (1:N)

هذه هي العلاقة الأكثر شيوعاً في المعلوماتية الحيوية. تعني أن كياناً واحداً في الجدول الأول يمكن أن يرتبط بعدة كيانات

في الجدول الثاني، لكن كل كيان في الجدول الثاني يرتبط بكيان واحد فقط في الجدول الأول.

This is the most common relationship in bioinformatics. It means one entity in the first table can relate to many entities in the second table, but each entity in the second table relates to only one entity in the first table.

أمثلة جوهرية في مشروعنا:

مثال 1: مريض → عينات (1:N)

مريض واحد يمكن أن يكون له عدة عينات مأخوذة في أوقات مختلفة أو من أنسجة مختلفة. كل عينة تنتمي لمريض واحد فقط.

Core examples in our project:

Example 1: Patient → Samples (1:N).

One patient can have several samples taken at different times or from different tissues. Each sample belongs to only one patient.

مثال 2: تجربة → تشغيلات (1:N)

تجربة واحدة (مثل مشروع RNA-seq كامل) قد تنتج عدة تشغيلات على جهاز التسلسل (لتغطية عمق أكبر). كل تشغيل ينتمي لتجربة واحدة.

Example 2: Experiment → Runs (1:N).

One experiment (e.g., a whole RNA-seq project) may produce several runs on the sequencer (for greater depth coverage). Each run belongs to one experiment.

مثال 3: جين → ميزات (1:N)

جين واحد له عدة إكسونات، وإنترونات، ومناطق تنظيمية. كل ميزة تنتمي لجين واحد فقط.

Example 3: Gene → Features (1:N)

One gene has several exons, introns, and regulatory regions. Each feature belongs to only one gene.

التنفيذ في SQL: نضع المفتاح الأجنبي (Foreign Key) في الجدول "المتعدد" (N)، والذي يشير إلى المفتاح الأساسي في الجدول "الواحد" (1).

Implementation in SQL: Place the Foreign Key in the "many" table (N), which points to the Primary Key in the "one" table (1).

SQL

```
-- الجدول "الواحد" (الآب)
CREATE TABLE genes (
    gene_id INTEGER PRIMARY KEY AUTOINCREMENT,
    symbol TEXT UNIQUE NOT NULL,
    description TEXT
);

-- الجدول "المتعدد" (الابن) - يحتوي على المفتاح الأجنبي
CREATE TABLE gene_features (
    feature_id INTEGER PRIMARY KEY AUTOINCREMENT,
    gene_id INTEGER NOT NULL, -- Foreign Key يشير إلى genes
    start INTEGER NOT NULL,
    end INTEGER NOT NULL,
    strand INTEGER NOT NULL CHECK(strand IN (1, -1)),
    feature_type TEXT NOT NULL,
    qualifiers TEXT, -- JSON
    FOREIGN KEY (gene_id) REFERENCES genes(gene_id) ON DELETE CASCADE
```

```
);

-- إنشاء فهرس لتسريع استعلامات الربط بين الجدولين
CREATE INDEX idx_features_gene_id ON gene_features(gene_id);
```

النوع الثالث: متعدد إلى متعدد (Many-to-Many — M:N)

Type Three: Many-to-Many (M:N).

هذه العلاقة تعكس أعقد حالات الترابط البيولوجي. تعني أن كياناً واحداً في الجدول الأول يمكن أن يرتبط بعدة كيانات في الجدول الثاني، وكيان واحد في الجدول الثاني يمكن أن يرتبط بعدة كيانات في الجدول الأول.

This relationship reflects the most complex cases of biological interconnection. It means one entity in the first table can relate to several entities in the second, and one entity in the second can relate to several entities in the first.

أهم مثال في المعلوماتية الحيوية: الجينات والمسارات الحيوية (Genes ↔️ Pathways)

· جين BRCA1 يشارك في مسار إصلاح DNA (Pathway A) ومسار موت الخلايا المبرمج (Pathway B).

· مسار إصلاح DNA (Pathway A) يحتوي على ATM ،BRCA2 ،BRCA1.

The most important example in bioinformatics: Genes ↔ Pathways.

· The BRCA1 gene participates in the DNA repair pathway (Pathway A) and the apoptosis pathway (Pathway B).

· The DNA repair pathway (Pathway A) contains BRCA1, BRCA2, and ATM.

لا يمكننا تمثيل هذه العلاقة في جدولين فقط، لأنها ستسبب تكراراً هائلاً. لو وضعنا pathway_id في جدول genes، لكان على BRCA1 أن يحتوي على معرفين (Pathway_A و Pathway_B) في

عمود واحد، وهذا ينتهك المبدأ الذري (Atomicity). الحل هو جدول وسيط (Junction Table أو Associative Table).

We cannot represent this relationship in just two tables, as it would cause massive redundancy. If we put pathway_id in the genes table, BRCA1 would have to contain two IDs (Pathway_A and Pathway_B) in one column, violating atomicity. The solution is a Junction Table (or Associative Table).

هذا الجدول الوسيط لا يمثل كياناً مستقلاً، بل يمثل العلاقة نفسها. يحتوي على مفتاحين أجنبيين: gene_id و pathway_id. كل صف في الجدول الوسيط يمثل ارتباطاً واحداً بين جين ومسار. يمكن أيضاً إضافة صفات للعلاقة (مثل confidence درجة الثقة في هذا الارتباط).

This Junction Table does not represent an independent entity, but rather the relationship itself. It contains two foreign keys: gene_id and pathway_id. Each row in the junction table represents one association between a gene and a

pathway. Attributes can also be added to the relationship (e.g., confidence score for this association).

SQL

```
-- جدول المسارات الحيوية --
CREATE TABLE pathways (
    pathway_id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT UNIQUE NOT NULL,
    description TEXT,
    source_db TEXT -- مثلا: KEGG, Reactome
);

-- الجدول الوسيط (Junction Table) لـ M:N
CREATE TABLE gene_pathways (
    gene_id INTEGER NOT NULL,
    pathway_id INTEGER NOT NULL,
    confidence REAL DEFAULT 1.0, -- درجة ثقة (0-1)
    PRIMARY KEY (gene_id, pathway_id), -- مفتاح مركب
    FOREIGN KEY (gene_id) REFERENCES genes(gene_id) ON DELETE CASCADE,
    FOREIGN KEY (pathway_id) REFERENCES pathways(pathway_id) ON DELETE CASCADE
);

-- فهارس لتسريع الاستعلام في كلا الاتجاهين --
CREATE INDEX idx_gp_gene ON gene_pathways(gene_id);
CREATE INDEX idx_gp_pathway ON gene_pathways(pathway_id);
```

ملاحظة هامة عن M:N في OOP مقابل SQL:

في OOP، قد يكون لديك Gene يحتوي على List[Pathway]، و Pathway يحتوي على List[Gene]. في SQL، هذا التداخل المتبادل يُفكك إلى جدولين أبويين وجدول وسيط. عند الاسترجاع، نستخدم JOIN مزدوج (أو استعلامين) لإعادة بناء القوائم في الذاكرة.

Important Note about M:N in OOP vs SQL:

In OOP, you might have Gene containing List[Pathway], and Pathway containing List[Gene]. In SQL, this mutual nesting is decomposed into two parent tables and one junction table. When retrieving, we use double JOIN (or two queries) to rebuild the lists in memory.

الفصل السادس: المفاتيح والقيود — هوية البيانات وسلامتها

Chapter Six: Keys and Constraints — Data Identity and Integrity

الشكل 6 — الخريطة المفاهيمية للفصل

Figure 6 — Conceptual map of the chapter

الشكل 6 — سير العمل الهندسي للفصل

Figure 6 — Engineering workflow of the chapter

الآن نعيد استخدام ما سبق، ولكن في سياق هندسي أوسع، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنتقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

This concept does not begin from zero; it begins with a skill you already have, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

6.5 المفاتيح والقيود مع مثال OOP — الهوية تنتقل من الكائن إلى السجل

6.5 Keys and Constraints with OOP — Identity Moves from Object to Record.

في Python قد تستخدم gene_id كخاصية داخل كائن. في SQL تصبح هذه الهوية جزءًا من العقد الذي يحمي الجدول. لذلك ينبغي أن يتطابق التصميم البرمجي مع قواعد البيانات دون أن يصبح أحدهما نسخة حرفية من الآخر.

In Python, gene_id may be an attribute of an object. In SQL, that identity becomes part of the contract that protects the table. The software model and the database model should therefore agree without becoming literal copies of one another.

Python

```
from dataclasses import dataclass


@dataclass(frozen=True)
class Gene:
    gene_id: int
    symbol: str
    organism: str
```

SQL

```
CREATE TABLE genes (
    gene_id INTEGER PRIMARY KEY,
    symbol TEXT NOT NULL UNIQUE,
    organism TEXT NOT NULL
);
```

بعد تحديد الكيانات والعلاقات، نحتاج إلى آليات تضمن تفرد كل سجل، وسلامة الروابط بين الجداول، وصحة القيم المدخلة. هذه الآليات تسمى القيود (Constraints)، وهي العمود الفقري لأي قاعدة بيانات موثوقة.

After identifying entities and relationships, we need mechanisms to ensure the uniqueness of each record, the integrity of links between tables, and the validity of entered values. These mechanisms are called Constraints, and they are the backbone of any reliable database.

أولاً: المفتاح الأساسي (Primary Key)

First: Primary Key

المفتاح الأساسي هو العمود (أو مجموعة الأعمدة) الذي يحدد كل صف في الجدول بشكل فريد. هو جوهر هوية البيانات. يجب أن يكون:

1. فريداً (Unique): لا يمكن أن يتكرر في صفين.
2. غير فارغ (NOT NULL): يجب أن يكون له قيمة دائماً.
3. ثابتاً (Stable): لا يتغير بمرور الوقت (لا نستخدم الاسم لأنه قد يتغير بالزواج، بل نستخدم معرفاً رقمياً).

The Primary Key is the column (or set of columns) that uniquely identifies each row in the table. It is the essence of data identity. It must be:

1. Unique: Cannot be duplicated in two rows.
2. NOT NULL: Must always have a value.
3. Stable: Does not change over time (we do not use a name as it might change with marriage; we use a numeric identifier).

أنواع المفاتيح الأساسية في الممارسة:

Types of Primary Keys in Practice:

1. المفتاح الطبيعي (Natural Key): معرف موجود في العالم الحقيقي، مثل رقم الانضمام في GenBank (NM_000660). مفيد لأنه معروف خارج النظام، لكنه قد يكون طويلاً أو يتغير (نادراً).
2. المفتاح الصناعي (Surrogate Key): معرف رقمي أو نصي يُنشأ داخل النظام، مثل INTEGER PRIMARY KEY AUTOINCREMENT. هذا هو الخيار المفضل في التصميم الهندسي، لأنه بسيط، سريع، ومستقل عن العالم الخارجي. المعرف الخارجي (مثل Accession Number) يُخزن كعمود UNIQUE منفصل.

3. Natural Key: An identifier existing in the real world, such as an accession number in GenBank (NM_000660). Useful because it is known outside the system, but it can be long or change (rarely).
4. Surrogate Key: A numeric or textual identifier generated inside the system, such as INTEGER PRIMARY KEY AUTOINCREMENT. This is the preferred

choice in engineering design, because it is simple, fast, and independent of the external world. The external identifier (e.g., Accession Number) is stored as a separate UNIQUE column.

ثانياً: المفتاح الأجنبي (Foreign Key)

Second: Foreign Key

المفتاح الأجنبي هو عمود (أو مجموعة أعمدة) في جدول يشير إلى المفتاح الأساسي في جدول آخر. هو الآلية التي نبني بها العلاقات (خاصة 1:N و M:N). دوره الأساسي هو الحفاظ على السلامة المرجعية (Referential Integrity).

The Foreign Key is a column (or set of columns) in a table that refers to the Primary Key in another table. It is the mechanism by which we build relationships (especially 1:N and M:N). Its primary role is to maintain Referential Integrity.

ماذا يعني هذا عملياً؟

· منع الإدخال غير الصحيح: لا يمكن إدخال sample بـ patient_id غير موجود في جدول patients.

· منع الحذف غير الآمن: لا يمكن حذف patient إذا كان هناك samples تشير إليه، إلا إذا استخدمنا ON DELETE CASCADE (الذي يحذف العينات تلقائياً) أو ON DELETE RESTRICT (الذي يمنع الحذف).

What does this mean in practice?

· Prevents incorrect insertion: You cannot insert a sample with a patient_id that does not exist in the patients table.

· Prevents unsafe deletion: You cannot delete a patient if there are samples pointing to it, unless you use ON DELETE CASCADE (which automatically deletes the samples) or ON DELETE RESTRICT (which blocks deletion).

SQL

```
-- مثال على مفتاح أجنبي مع ON DELETE CASCADE
-- Example of a Foreign Key with ON DELETE CASCADE
CREATE TABLE samples (
    sample_id INTEGER PRIMARY KEY AUTOINCREMENT,
    patient_id INTEGER NOT NULL,
    tissue_type TEXT,
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id) ON DELETE CASCADE
);
```

ثالثاً: القيد الفريد (Unique Constraint) وقيود التحقق (Check Constraints)

Third: Unique Constraint and Check Constraints.

- UNIQUE: يمنع تكرار القيم في عمود معين (أو مجموعة أعمدة). يُستخدم للمعرفات الخارجية (Accession Numbers) أو أسماء الجينات (Symbols) التي يجب أن تكون فريدة.

· CHECK: يضمن أن القيم في عمود معين تتبع شرطاً منطقياً. مثلاً، sex IN ('M', 'F', 'O')، age >= 0، أو strand IN (1, -1). هذا يمنع إدخال بيانات غير منطقية على مستوى قاعدة البيانات، قبل وصولها إلى طبقة التطبيق.

· UNIQUE: Prevents duplication of values in a specific column (or set of columns). Used for external identifiers (Accession Numbers) or gene symbols that must be unique.

· CHECK: Ensures that values in a specific column follow a logical condition. For example, age >= 0, sex IN ('M', 'F', 'O'), or strand IN (1, -1). This prevents illogical data from being entered at the database level, before reaching the application layer.

SQL

```sql
CREATE TABLE genes (
   gene_id INTEGER PRIMARY KEY AUTOINCREMENT,
   symbol TEXT UNIQUE NOT NULL,  -- لا يمكن تكرار رمز جين
   chromosome TEXT CHECK(chromosome GLOB '[0-9XY]*'), -- تحقق بسيط من الصيغة
   gc_content REAL CHECK(gc_content BETWEEN 0 AND 100)
);
```

رابعاً: لماذا لا يجوز الاعتماد على ترتيب الصفوف؟

Fourth: Why Can We Not Rely on Row Order?

في ملفات CSV، قد تعتمد على أن الصف رقم 5 هو العينة S005. في قاعدة البيانات، هذا الافتراض خاطئ تماماً. محرك قاعدة البيانات لا يضمن ترتيب الصفوف إلا إذا طلبت ذلك صراحةً باستخدام ORDER BY. قد يعيد الصفوف بترتيب حسب المفتاح الأساسي، أو حسب الفهرس المستخدم، أو حسب خطة التنفيذ التي يختارها المحرك لتحسين الأداء. الاعتماد على الترتيب هو وصفة لكارثة صامتة: البرنامج يعمل، لكنه يربط البيانات الخاطئة مع بعضها.

In CSV files, you might rely on row number 5 being sample S005. In a database, this assumption is completely false. The database engine does not guarantee row order unless you explicitly request it using ORDER BY. It might

return rows ordered by the primary key, or by the index used, or by the execution plan chosen by the engine for performance optimization. Relying on order is a recipe for silent disaster: the program runs, but it links the wrong data together.

الحل: استخدم دائماً المعرفات الفريدة (Primary Keys) للرجوع إلى الصفوف. إذا أردت تكراراً أو ترتيباً معيناً، استخدم ORDER BY مع عمود زمني (مثل created_at) أو عمود ترتيبي (مثل position). لا تثق أبداً في الترتيب الضمني.

Solution: Always use unique identifiers (Primary Keys) to refer to rows. If you need a specific order or repeatability, use ORDER BY with a timestamp column (like created_at) or an ordinal column (like position). Never trust implicit order.

الفصل السابع: تصميم المخطط (Schema Design) — مشروع Gene Resistance Tracker

Chapter Seven: Schema Design — The Gene Resistance Tracker Project

يجب أن تستطيع رسم دورة حياة Gene Resistance Tracker، قبل أن تكتب أول استعلام في مشروع المعلومة: من العينة إلى العزلة أو السلالة، ثم الجين أو المتغير، ثم اختبار المقاومة، ثم النتيجة، ثم مصدر البيانات وإصدارها. هذا الرسم ليس توثيقًا تجميليًا؛ إنه وسيلة لاكتشاف المفاتيح المفقودة والعلاقات المتعددة والتكرار قبل أن يتحول إلى مشكلة في SQL. كما يجب أن تميّز بين «القيمة المقاسة» و«التفسير» و«المصدر» حتى لا يصبح القرار العلمي جزءًا غير قابل للتتبع من سجل واحد.

Before writing the first query for the Gene Resistance Tracker, you should be able to draw the lifecycle of information: from sample to isolate or strain, then to gene or variant, then to the resistance assay, result, and data provenance/version. This model is not decorative documentation; it is a way to discover missing keys, many-to-many relationships, and redundancy before they become SQL problems. You should also distinguish the measured value from its interpretation and its source, so that the scientific decision does not become an untraceable field buried inside a single record.

مختبر: مخطط متكامل لمقاومة المضادات الحيوية

Laboratory: A Complete Resistance Schema

يصبح المخطط مفيدًا عندما يجيب كل جدول عن سؤال مستقل ويعبّر كل مفتاح عن علاقة حقيقية. أبقِ دليل ارتباط الطفرة بالدواء داخل جدول ربط لأن العلاقة نفسها تحمل الثقة والمصدر.

A schema becomes useful when every table answers a distinct question and every key expresses a real relationship. Keep mutation-drug evidence in a junction table because the relationship itself carries confidence and provenance.

يجب تصميم المخطط بحيث يستطيع التقرير اللاحق إعادة بناء الملاحظات الدقيقة التي تقف وراء ادعاء المقاومة.

The schema should be designed so that a later report can reconstruct the exact observations behind a resistance claim.

نمذجة علاقة many-to-many بيولوجية

Modeling a many-to-many biological relation

```sql
CREATE TABLE mutation_drug (
    mutation_id INTEGER NOT NULL,
    drug_id INTEGER NOT NULL,
    evidence_level TEXT NOT NULL,
    confidence REAL NOT NULL CHECK(confidence BETWEEN 0.0 AND 1.0),
    PRIMARY KEY (mutation_id, drug_id),
    FOREIGN KEY (mutation_id) REFERENCES mutations(mutation_id),
    FOREIGN KEY (drug_id) REFERENCES drugs(drug_id)
);
```

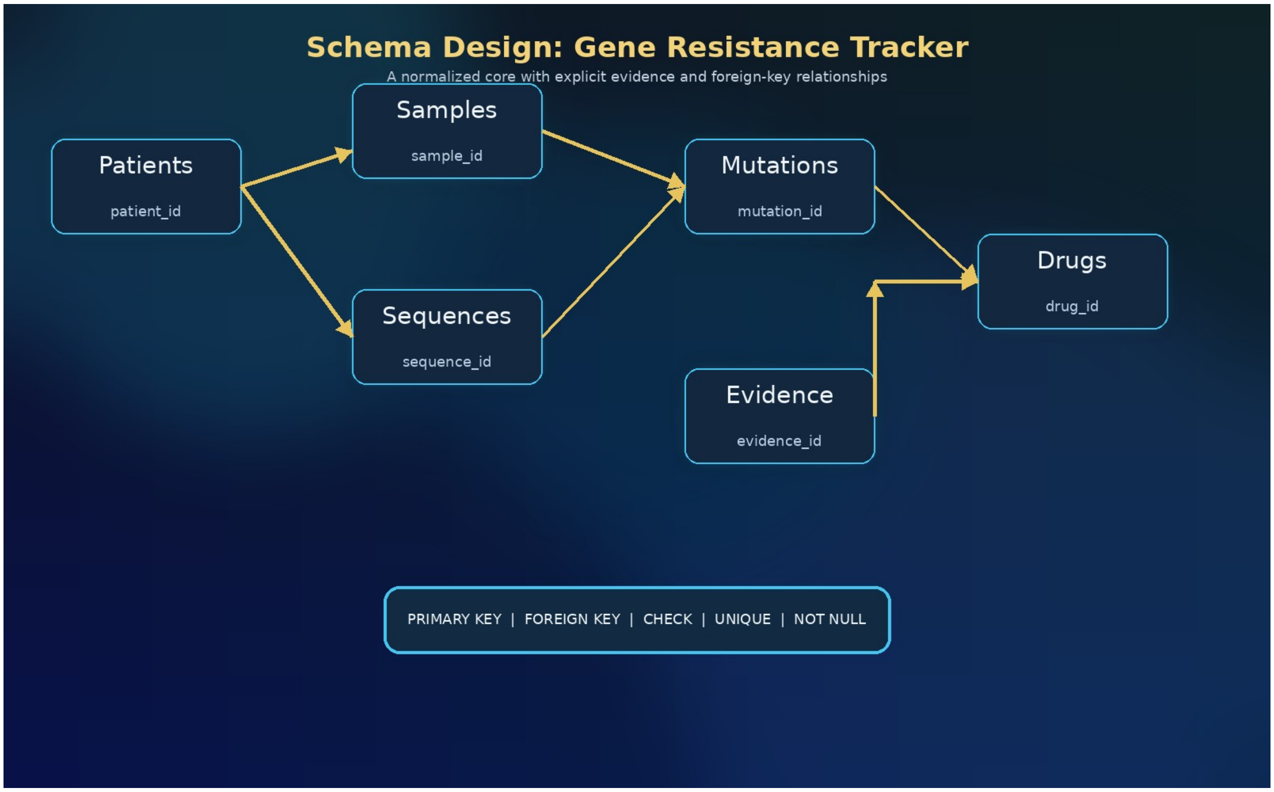
SQL

الشكل 7 — الخريطة المفاهيمية للفصل

Figure 7 — Conceptual map of the chapter

الشكل 7 — سير العمل الهندسي للفصل

Figure 7 — Engineering workflow of the chapter



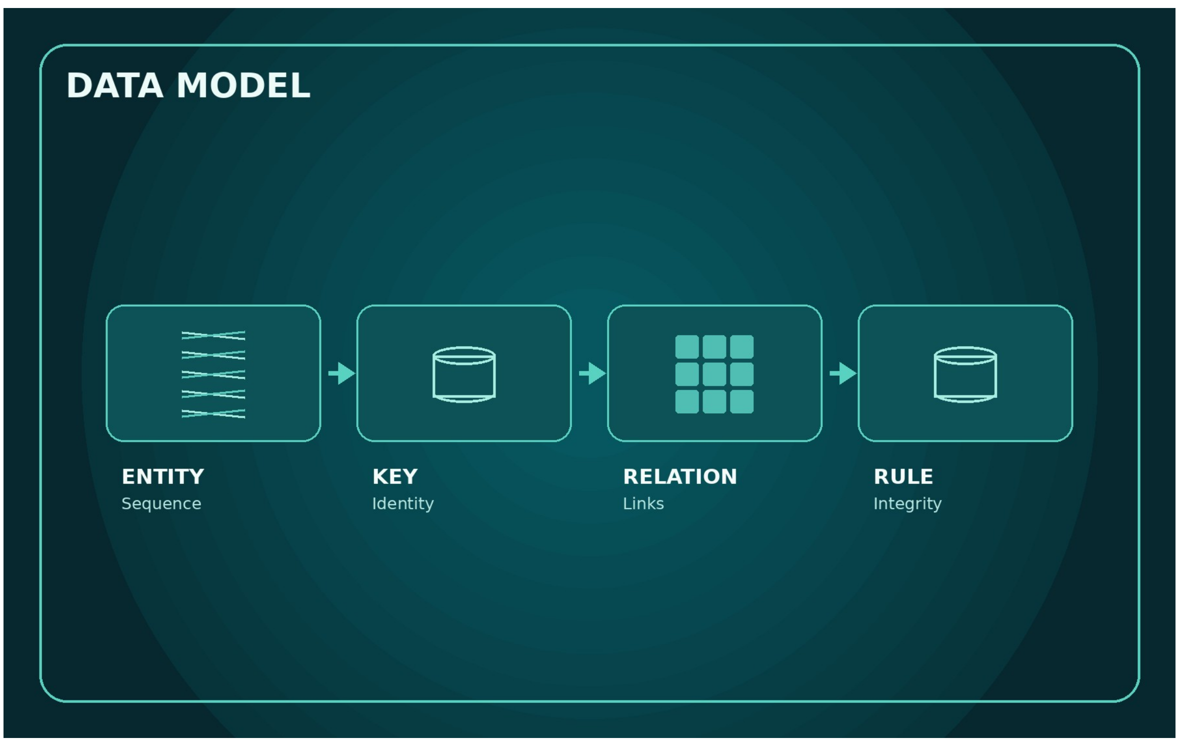
من السؤال العلمي إلى الجداول والمفاتيح والقيود ومسارات البحث.

From the scientific question to tables, keys, constraints, and efficient access paths.

المفاهيم التي بُنيت في الجزء الثاني تأخذ هنا وظيفة جديدة، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

We now reuse an earlier idea, but in a broader engineering context, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

7.5 From Entity to Table — Step-by-Step Data Modeling.



7.5 من الكيان إلى الجدول — نمذجة البيانات خطوة بخطوة

Figure — DATA MODEL

ابدأ بالسؤال العلمي، ثم حدّد الأشياء التي تحتاج إلى وجود مستقل داخل النظام. هذه الأشياء تصبح كيانات. بعد ذلك حدّد خصائص كل كيان، ثم اختر هوية مستقرة لكل سجل، ثم ارسم العلاقات، وأخيرًا حوّل النموذج إلى جداول وقيود. بهذا الترتيب لا تبدأ من SQL؛ بل تجعل SQL نتيجة لنموذج منطقي سابق.

Start with the scientific question, then identify the things that must exist independently in the system. These become entities. Next identify their attributes, choose a stable identity for each record, draw the relationships, and only then translate the model into tables and constraints. In this order, you do not start with SQL; SQL becomes the implementation of a prior logical model.

في مشروع تتبع مقاومة الجينات، يمكن أن تكون الكيانات الأساسية Patient وSample وSequence وGene وMutation

وResistanceObservation. لا تضع كل هذه المعلومات في جدول واحد؛ لأن التكرار سيجعل تحديث البيانات عرضة للتناقض. افصل الكيانات التي لها هوية مستقلة، ثم اربطها بالمفاتيح الخارجية.

In a gene-resistance tracking project, core entities may include Patient, Sample, Sequence, Gene, Mutation, and ResistanceObservation. Do not place all of this information in one table, because duplication makes updates vulnerable to inconsistency. Separate entities that have independent identities and connect them through foreign keys.

SQL

```sql
CREATE TABLE genes (
    gene_id INTEGER PRIMARY KEY,
    symbol TEXT NOT NULL UNIQUE,
    organism TEXT NOT NULL
);

CREATE TABLE sequences (
    sequence_id INTEGER PRIMARY KEY,
    gene_id INTEGER,
    accession TEXT NOT NULL UNIQUE,
    sequence TEXT NOT NULL,
    FOREIGN KEY (gene_id) REFERENCES genes(gene_id)
);
```

يمنحك هذا النموذج نقطة انطلاق قابلة للتوسع: يمكن إضافة نسخ بروتينية أو تشغيلات أو إصدارات دون إعادة تعريف معنى الجين نفسه. وتدعم SQLite صراحةً PRIMARY KEY وUNIQUE وNOT NULL وCHECK وFOREIGN KEY ضمن تعريف الجدول.

This model gives you an extensible starting point: protein products, runs, or versions can be added without redefining what a gene means. SQLite explicitly supports PRIMARY KEY, UNIQUE, NOT NULL, CHECK, and FOREIGN KEY constraints in table definitions.

الآن وقد فهمنا الكيانات، والعلاقات، والمفاتيح، والقيود، حان الوقت لتطبيق كل هذا في مخطط متكامل. سنصمم مخططاً لمشروعنا الهندسي الموحد: "Gene Resistance Tracker"، الذي سيرافقنا خلال بقية أبواب الجزء الثالث.

Now that we understand entities, relationships, keys, and constraints, it is time to apply all of this in an integrated Schema. We will design a Schema for

our unified engineering project: "Gene Resistance Tracker", which will accompany us through the rest of Part Three.

الغرض من المشروع: تتبع عينات من المرضى، وتحليل تسلسلات جينات مقاومة المضادات الحيوية فيها، واكتشاف الطفرات المرتبطة بالمقاومة، وربط هذه الطفرات بالأدوية ودرجات الثقة.

Project Purpose: To track patient samples, analyze the sequences of antibiotic resistance genes within them, detect mutations associated with resistance, and link these mutations to drugs and confidence scores.

المتطلبات (Requirements):

1. لكل مريض: معرف خارجي (مجهول)، عمر، جنس، تشخيص.

2. لكل عينة: تاريخ الجمع، نوع النسيج، جودة الـ RNA، الحالة (مريض/سليم).

3. كل عينة لها تسلسل واحد أو أكثر (جينات مختلفة).

4. كل تسلسل يحمل اسم الجين، النص الكامل للـ DNA، الطول، ونسبة GC.

5. لكل تسلسل، نحدد طفرات (Mutations) في مواقع محددة (تبديل قاعدة بأخرى).

6. كل طفرة مرتبطة بمقاومة دواء واحد أو أكثر (علاقة M:N).

7. لكل ربط بين طفرة ومقاومة، هناك درجة ثقة (0-1).

8. يجب تتبع من أدخل البيانات ومتى (Audit Log).

المخطط النهائي (Final Schema):

SQL

```sql
-- ============================================================
```

```sql
-- 1. جدول المرضى (Patients)
-- ============================================================
CREATE TABLE patients (
    patient_id INTEGER PRIMARY KEY AUTOINCREMENT,
    external_id TEXT UNIQUE NOT NULL,  -- معرف مجهول للربط الخارجي
    age INTEGER CHECK(age >= 0 AND age < 150),
    sex TEXT CHECK(sex IN ('M', 'F', 'O')),
    diagnosis TEXT,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);

-- ============================================================
-- 2. جدول العينات (Samples)
-- ============================================================
CREATE TABLE samples (
    sample_id INTEGER PRIMARY KEY AUTOINCREMENT,
    patient_id INTEGER NOT NULL,
    tissue_type TEXT NOT NULL,
    collection_date TEXT,
    rna_quality REAL CHECK(rna_quality BETWEEN 0 AND 10), -- RIN score
    status TEXT CHECK(status IN ('control', 'diseased', 'treated')),
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id) ON DELETE CASCADE
);

-- فهرس لتسريع البحث عن عينات مريض معين
CREATE INDEX idx_samples_patient ON samples(patient_id);

-- ============================================================
-- 3. جدول التسلسلات (Sequences)
-- ============================================================
CREATE TABLE sequences (
```

```sql
    sequence_id INTEGER PRIMARY KEY AUTOINCREMENT,
    sample_id INTEGER NOT NULL,
    gene_symbol TEXT NOT NULL, -- مثل "mecA", "KPC"
    sequence_text TEXT NOT NULL, -- النص الطويل للـ DNA
    length INTEGER,
    gc_content REAL CHECK(gc_content BETWEEN 0 AND 100),
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sample_id) REFERENCES samples(sample_id) ON DELETE CASCADE
);

CREATE INDEX idx_sequences_sample ON sequences(sample_id);
CREATE INDEX idx_sequences_gene ON sequences(gene_symbol);

-- ============================================================
-- 4. جدول الطفرات (Mutations)
-- ============================================================
CREATE TABLE mutations (
    mutation_id INTEGER PRIMARY KEY AUTOINCREMENT,
    sequence_id INTEGER NOT NULL,
    position INTEGER NOT NULL,  -- الموقع على التسلسل (1-based)
    ref_base TEXT NOT NULL CHECK(ref_base IN ('A', 'T', 'C', 'G')),
    alt_base TEXT NOT NULL CHECK(alt_base IN ('A', 'T', 'C', 'G')),
    effect TEXT,  -- "missense", "nonsense", "silent"
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sequence_id) REFERENCES sequences(sequence_id) ON DELETE CASCADE,
    UNIQUE (sequence_id, position)  -- لا يمكن تكرار طفرة في نفس الموقع لنفس التسلسل
);

CREATE INDEX idx_mutations_sequence ON mutations(sequence_id);

-- ============================================================
-- 5. جدول الأدوية والمقاومة (Resistance)
```

```sql
-- ============================================================
CREATE TABLE resistance (
    resistance_id INTEGER PRIMARY KEY AUTOINCREMENT,
    drug_name TEXT UNIQUE NOT NULL,  -- مثل "Methicillin", "Ciprofloxacin"
    drug_class TEXT,  -- مثل "Beta-lactam", "Fluoroquinolone"
    description TEXT
);

-- ============================================================
-- 6. الجدول الوسيط: الطفرات ←→ المقاومة (M:N)
-- ============================================================
CREATE TABLE mutation_resistance (
    mutation_id INTEGER NOT NULL,
    resistance_id INTEGER NOT NULL,
    confidence REAL DEFAULT 1.0 CHECK(confidence BETWEEN 0 AND 1),
    evidence TEXT,  -- رابط للدراسة أو المرجع
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (mutation_id, resistance_id),
    FOREIGN KEY (mutation_id) REFERENCES mutations(mutation_id) ON DELETE CASCADE,
    FOREIGN KEY (resistance_id) REFERENCES resistance(resistance_id) ON DELETE CASCADE
);

-- فهارس للربط السريع
CREATE INDEX idx_mr_mutation ON mutation_resistance(mutation_id);
CREATE INDEX idx_mr_resistance ON mutation_resistance(resistance_id);

-- ============================================================
-- 7. جدول المستخدمين (للأمن والمراجعة)
-- ============================================================
CREATE TABLE users (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE NOT NULL,
```

```
    role TEXT CHECK(role IN ('admin', 'researcher', 'auditor')) DEFAULT 'researcher',
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);

-- ============================================================
-- 8. جدول سجل التدقيق (Audit Log) - تتبع كل عملية حساسة
-- ============================================================
CREATE TABLE audit_log (
    audit_id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    action TEXT NOT NULL,  -- 'INSERT', 'UPDATE', 'DELETE', 'SELECT'
    table_name TEXT NOT NULL,
    record_id TEXT,  -- المعرف المسجل (قد يكون رقمي أو نصي)
    old_value TEXT,  -- JSON للقيمة القديمة (اختياري)
    new_value TEXT,  -- JSON للقيمة الجديدة (اختياري)
    timestamp TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE SET NULL
);

-- فهرس للبحث السريع في سجلات التدقيق حسب الجدول والزمن
CREATE INDEX idx_audit_table ON audit_log(table_name);
CREATE INDEX idx_audit_time ON audit_log(timestamp);
```

تحليل المخطط:

Schema Analysis:

1. التطبيع الكامل: كل حقيقة في مكان واحد. بيانات المريض منفصلة عن العينة، والعينة عن التسلسل، والتسلسل عن الطفرة.

2. العلاقات واضحة: 1:N عبر المفاتيح الأجنبية (مثل sample_id في sequences). M:N عبر الجدول الوسيط mutation_resistance.

3. القيود تحمي البيانات: CHECK يمنع الأعمار السالبة ونسب GC الخارجة عن النطاق. UNIQUE يمنع تكرار معرف المريض الخارجي أو تكرار طفرة في نفس الموقع.

4. الكفاءة: الفهارس (Indexes) على المفاتيح الأجنبية تسرع استعلامات JOIN بشكل كبير.

6. Full Normalization: Every fact is in one place. Patient data is separate from samples, samples from sequences, sequences from mutations.

7. Clear Relationships: 1:N via foreign keys (e.g., sample_id in sequences). M:N via the junction table mutation_resistance.

8. Constraints Protect Data: CHECK prevents negative ages and out-of-range GC percentages. UNIQUE prevents duplicate external patient IDs or duplicate mutations at the same position.

5. الأمن والمراجعة: audit_log و users يسمحان بتتبع من قام بأي تغيير، وهو مبدأ أساسي من biobrick_security في الجزء الثاني.

9. Efficiency: Indexes on foreign keys significantly speed up JOIN queries.

10. Security and Auditability: audit_log and users enable tracking who made what change, a fundamental principle from biobrick_security in Part Two.

ربط المخطط بـ OOP (الجزء العملي القادم):

Connecting the Schema to OOP (The Next Practical Step):

هذا المخطط ليس مجرد SQL؛ إنه مرآة لتصميم OOP. كل جدول سيكون له Repository خاص به (مثل PatientRepository، MutationRepository، SampleRepository). هذه الـ Repositories ستستخدم sqlite3 لتنفيذ عمليات CRUD (Create, Read, Update,

Delete)، وستحول الصفوف إلى كائنات @dataclass والعكس. العلاقة M:N بين الطفرات والمقاومة ستتم إدارتها عبر MutationResistanceRepository الذي يتعامل مع الجدول الوسيط.

This Schema is not just SQL; it is a mirror of OOP design. Each table will have its own Repository (e.g., PatientRepository, SampleRepository, MutationRepository). These Repositories will use sqlite3 to perform CRUD operations (Create, Read, Update, Delete), and convert rows to @dataclass objects and vice versa. The M:N relationship between mutations and resistance will be managed via a MutationResistanceRepository that handles the junction table.

مثال على استعلام علمي من هذا المخطط:

Example of a Scientific Query from this Schema:

السؤال: "أعطني جميع المرضى الذين لديهم طفرة في جين mecA مرتبطة بمقاومة الميثيسيلين بدرجة ثقة أكبر من 0.8."

Question: "Give me all patients who have a mutation in the mecA gene associated with methicillin resistance with a confidence greater than 0.8."

SQL

```
SELECT DISTINCT p.external_id, p.age, m.position, m.alt_base, mr.confidence
FROM patients p
JOIN samples s ON p.patient_id = s.patient_id
JOIN sequences seq ON s.sample_id = seq.sample_id
JOIN mutations m ON seq.sequence_id = m.sequence_id
JOIN mutation_resistance mr ON m.mutation_id = mr.mutation_id
JOIN resistance r ON mr.resistance_id = r.resistance_id
WHERE seq.gene_symbol = 'mecA'
  AND r.drug_name = 'Methicillin'
  AND mr.confidence > 0.8;
```

لاحظ كيف يعبر هذا الاستعلام عن المنطق البيولوجي مباشرة، دون أي حلقات برمجية معقدة. هذا هو هدف التصميم العلاقي: جعل الأسئلة العلمية قابلة للترجمة إلى SQL واضحة وقابلة للتنفيذ بكفاءة.

Note how this query expresses biological logic directly, without any complex programming loops. This is the goal of relational design: making scientific questions translatable into clear, efficiently executable SQL.

الانتقال إلى الباب الثالث:

Transition to Unit Three:

الآن أصبح المخطط جاهزاً. لدينا خريطة كاملة للبيانات: الكيانات، والعلاقات، والمفاتيح، والقيود. في الباب الثالث، سنتعلم كيف نترجم هذا المخطط إلى استعلامات SQL فعلية. سنبدأ بـ SELECT البسيط، ثم WHERE للتصفية، ثم JOIN لربط الجداول (وهو ما سيكون سهلاً الآن لأننا فهمنا العلاقات مسبقاً)، ثم GROUP BY و HAVING للتجميعات، وأخيراً CTEs للاستعلامات المعقدة. الباب الثالث سيكون تطبيقاً عملياً مباشراً على المخطط الذي صممناه هنا.

Now the Schema is ready. We have a complete map of the data: entities, relationships, keys, and constraints. In Unit Three, we will learn how to translate this Schema into actual SQL queries. We will start with simple SELECT, then WHERE for filtering, then JOIN to link tables (which will be easy now because we already understand the relationships), then GROUP BY and HAVING for aggregations, and finally CTEs for complex queries. Unit Three will be a direct practical application of the Schema we designed here.

الباب الثالث: **SQL** من الصفر إلى الاستعلام العلمي

Unit Three: SQL from Zero to Scientific Query

الغرض من هذا الباب:

تحويل المخطط (Schema) الذي صممناه في الباب الثاني إلى استعلامات SQL فعلية. نبدأ من أبسط استعلامات القراءة، ونتدرج إلى التصفية، والترتيب، والتجميع، والربط بين الجداول (JOIN)، والاستعلامات المتداخلة (Subqueries) وعبارات CTE، لننتهي بتطبيق SQL كأداة للإجابة عن الأسئلة العلمية مباشرة، تماماً كما كنا نفعل مع دوال Python في الجزء الثاني.

Purpose of this unit:

To transform the Schema designed in Unit Two into actual SQL queries. We start from the simplest read queries, progress to filtering, ordering, grouping, joining tables (JOIN), subqueries and CTEs, and finally apply SQL as a tool to directly answer scientific questions, just as we did with Python functions in Part Two.

الفصل الثامن: SELECT / WHERE / ORDER BY — اللغة الأولى للاستعلام

Chapter Eight: SELECT / WHERE / ORDER BY — The First Language of Querying

مختبر: استرجاع عينة مدروسة

Laboratory: Controlled Cohort Retrieval

يبدأ SELECT العلمي بتحديد المجتمع المدروس. حدّد الكائن والفترة الزمنية وحد الجودة والحقول المطلوبة للتحليل قبل كتابة الاستعلام.

A scientific SELECT begins with a defined population. State the organism, time window, quality threshold, and fields required by the analysis before writing the query.

أعد الحقول المطلوبة في المرحلة اللاحقة فقط. يجعل عقد النتيجة الضيق الكود أسهل في الاختبار والاستعلام أسهل في التدقيق.

Return only the fields required downstream. A narrow result contract makes the code easier to test and the query easier to audit.

استعلام عينة بمعلمات

Parameterized cohort query

```
SELECT s.sample_id, s.collection_date, s.read_count
FROM samples AS s
WHERE s.organism = ?
  AND s.collection_date >= ?
  AND s.read_count >= ?
ORDER BY s.collection_date, s.sample_id;
```

SQL

الشكل 8 —— الخريطة المفاهيمية للفصل

Figure 8 — Conceptual map of the chapter

الشكل 8 —— سير العمل الهندسي للفصل

Figure 8 — Engineering workflow of the chapter

من منظور المسار الكامل للكتاب، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

The concepts developed in Part Two take on a new role here, you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

8.5 SELECT مع Repository — الاستعلام يصبح خدمة قابلة لإعادة الاستخدام

8.5 SELECT with a Repository — Turning a Query into a Reusable Service.

لا تضع اتصال قاعدة البيانات داخل كل مكان في البرنامج. اجعل Repository مسؤولًا عن الوصول إلى البيانات، ثم حوّل الصفوف إلى كائنات ذات معنى علمي.

Do not scatter database connections throughout the program. Let a Repository own data access, then convert rows into objects with scientific meaning.

Python

```
from dataclasses import dataclass
import sqlite3


@dataclass
class GeneRow:
    gene_id: int
    symbol: str


class GeneRepository:
```

```
    def __init__(self, db_path: str):
        self.db_path = db_path

    def all_genes(self) -> list[GeneRow]:
        with sqlite3.connect(self.db_path) as conn:
            rows = conn.execute(
                "SELECT gene_id, symbol FROM genes ORDER BY symbol"
            ).fetchall()
        return [GeneRow(*row) for row in rows]
```

بعد أن صممنا المخطط في الباب السابق، حان الوقت لاستجواب البيانات. اللغة الأولى التي سيتعلمها القارئ هي SELECT. لكنها ليست مجرد كلمة؛ إنها تعبير عن الرغبة في رؤية جزء معين من الواقع المخزن في الجداول. أبسط صورها هو جلب كل شيء من جدول، لكن القوة الحقيقية تظهر عند التصفية (WHERE) والترتيب (ORDER BY).

After designing the Schema in the previous unit, it is time to interrogate the data. The first language the reader will learn is SELECT. However, it is not just a keyword; it is an expression of the desire to see a specific part of the reality

stored in the tables. Its simplest form retrieves everything from a table, but the real power emerges with filtering (WHERE) and ordering (ORDER BY).

8.1 الاستعلام الأساسي: SELECT و DISTINCT

8.1 Basic Query: SELECT and DISTINCT.

الاستعلام الأبسط هو جلب جميع الأعمدة من جدول معين.

في مشروعنا، لنبدأ بجدول patients:

The simplest query retrieves all columns from a specific table. In our project, let us start with the patients table:

SQL

```
-- جلب جميع بيانات المرضى
-- Retrieve all patient data
SELECT * FROM patients;
```

لكن في التطبيقات الواقعية، نادراً ما نريد كل الأعمدة، خاصة إذا كانت تحتوي على نصوص طويلة أو بيانات حساسة. نحدد الأعمدة المطلوبة بالاسم:

However, in real applications, we rarely want all columns, especially if they contain long texts or sensitive data. We specify the required columns by name:

SQL

```
-- جلب المعرف الخارجي والعمر والتشخيص فقط، مع تجاهل الأعمدة الداخلية
-- Retrieve only external ID, age, and diagnosis, ignoring internal columns
SELECT external_id, age, diagnosis FROM patients;
```

أحياناً نحتاج إلى رؤية القيم الفريدة في عمود معين، مثل قائمة الأمراض المختلفة المسجلة. هنا يأتي دور DISTINCT:

Sometimes we need to see the unique values in a specific column, such as the list of different recorded diseases. This is where DISTINCT comes in:

SQL

```sql
-- جلب قائمة فريدة بكل التشخيصات المسجلة
-- Retrieve a unique list of all recorded diagnoses
SELECT DISTINCT diagnosis FROM patients;
```

8.2 التصفية المنطقية: WHERE

8.2 Logical Filtering: WHERE.

WHERE هو الفاصل بين عرض كل شيء وعرض ما يهمنا فقط. هو أداة التصفية الأساسية. في WHERE، نستخدم عوامل المقارنة (=, >, <, >=, <=, <>) والعوامل المنطقية (AND, OR, NOT). والأداة الأكثر مرونة هي LIKE للبحث عن أنماط نصية، و IN للبحث ضمن قائمة.

WHERE is the boundary between showing everything and showing only what matters. It is the basic filtering tool. In WHERE, we use comparison operators (=, >, <, >=, <=, <>) and logical operators (AND, OR, NOT). The most flexible tools are LIKE for text pattern matching and IN for searching within a list.

مثال 1: فلترة رقمية بسيطة (Simple Numerical Filter):

جلب المرضى الذين تزيد أعمارهم عن 50 عاماً.

Retrieve patients older than 50.

SQL

```
SELECT external_id, age, diagnosis
FROM patients
WHERE age > 50;
```

مثال 2: فلترة نصية بالضبط (Exact Text Filter):

جلب المرضى الذين تم تشخيصهم بسرطان الثدي.

Retrieve patients diagnosed with breast cancer.

SQL

```
SELECT external_id, age, diagnosis
FROM patients
WHERE diagnosis = 'breast_cancer';
```

جلب المرضى الذين تتراوح أعمارهم بين 40 و 60 عاماً ومصابين بسرطان الثدي.

مثال 3: فلترة مركبة (AND) (Compound Filter - AND):

Retrieve patients aged between 40 and 60 with breast cancer.

SQL

```
SELECT external_id, age, diagnosis
FROM patients
WHERE age BETWEEN 40 AND 60
  AND diagnosis = 'breast_cancer';
```

جلب الجينات التي تبدأ بـ "BRCA" (رمز الجين BRCA1، BRCA2، إلخ).

مثال 4: البحث عن نمط نصي (LIKE) (Text Pattern Search - LIKE):

Retrieve genes whose symbol starts with "BRCA".

SQL

```
SELECT gene_id, gene_symbol, description
FROM sequences
WHERE gene_symbol LIKE 'BRCA%';
```

ملاحظة: % تعني أي عدد من الأحرف، بينما _ تعني حرفاً واحداً.

Note: % means any number of characters, while _ means a single character.

جلب المرضى الذين لديهم أحد تشخيصين محددين.

مثال 5: البحث ضمن قائمة (IN) (Search within a List - IN):

Retrieve patients who have one of two specific diagnoses.

SQL

```
SELECT external_id, age, diagnosis
FROM patients
WHERE diagnosis IN ('breast_cancer', 'lung_cancer');
```

8.3 الترتيب: ORDER BY

8.3 Ordering: ORDER BY

البيانات التي تُرجعها قاعدة البيانات ليست مرتبة افتراضياً.

ORDER BY يسمح لنا بفرز النتائج تصاعدياً (ASC) أو تنازلياً (DESC) بناءً على عمود واحد أو عدة أعمدة.

The data returned by the database is not ordered by default. ORDER BY allows us to sort the results ascending (ASC) or descending (DESC) based on one or multiple columns.

مثال:

جلب المرضى الأكبر سناً أولاً.

Retrieve the oldest patients first.

SQL

```
SELECT external_id, age, diagnosis
FROM patients
WHERE age IS NOT NULL
```

```
ORDER BY age DESC;
```

مثال مع أعمدة متعددة:

جلب العينات مرتبة حسب المريض أولاً (تصاعدي)، ثم حسب تاريخ الجمع (تنازلياً، أي الأحدث أولاً).

Example with multiple columns:

Retrieve samples ordered by patient first (ascending), then by collection date (descending, i.e., most recent first).

SQL

```
SELECT sample_id, patient_id, tissue_type, collection_date
FROM samples
ORDER BY patient_id ASC, collection_date DESC;
```

8.4 ربط SELECT بـ OOP (عملي)

8.4 Connecting SELECT to OOP (Practical).

في GeneRepository الذي بنيناه في الباب الأول، قمنا بتنفيذ find_by_id. الآن، دعنا نضيف دوال استعلام أكثر مرونة باستخدام WHERE و ORDER BY، مع إعادة النتائج ككائنات OOP.

In the GeneRepository we built in Unit One, we implemented find_by_id. Now, let's add more flexible query functions using WHERE and ORDER BY, returning results as OOP objects.

Python

```
import sqlite3
from typing import List, Optional
from dataclasses import dataclass


# تعريف كائنات OOP (مبسطة للعرض، مع إعادة استخدام Patient و Sample من الباب الثاني)
# Define OOP objects (simplified for display, reusing Patient and Sample from Unit Two)
@dataclass
class Patient:
    patient_id: int
    external_id: str
    age: Optional[int] = None
    sex: Optional[str] = None
    diagnosis: Optional[str] = None

@dataclass
class Sequence:
```

```
    sequence_id: int
    sample_id: int
    gene_symbol: str
    sequence_text: str
    length: Optional[int] = None
    gc_content: Optional[float] = None


class PatientRepository:
    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path


    def find_by_age_range(self, min_age: int, max_age: int) -> List[Patient]:
        """استرجاع المرضى في نطاق عمري معين، مرتبة تنازلياً حسب العمر.
        Retrieve patients within a specific age range, ordered descending by age.
        """
        query = """
            SELECT patient_id, external_id, age, sex, diagnosis
            FROM patients
            WHERE age >= ? AND age <= ?
            ORDER BY age DESC
        """
        with sqlite3.connect(self.db_path) as conn:
            conn.row_factory = sqlite3.Row
            rows = conn.execute(query, (min_age, max_age)).fetchall()
            return [
                Patient(
                    patient_id=r["patient_id"],
                    external_id=r["external_id"],
                    age=r["age"],
                    sex=r["sex"],
                    diagnosis=r["diagnosis"]
                ) for r in rows
```

```
        ]

    def find_by_diagnosis(self, diagnosis: str) -> List[Patient]:
        """استرجاع المرضى بتشخيص محدد، مرتبة أبجدياً حسب المعرف الخارجي.
        Retrieve patients with a specific diagnosis, ordered alphabetically by external ID.
        """
        query = """
            SELECT patient_id, external_id, age, sex, diagnosis
            FROM patients
            WHERE diagnosis = ?
            ORDER BY external_id ASC
        """
        with sqlite3.connect(self.db_path) as conn:
            conn.row_factory = sqlite3.Row
            rows = conn.execute(query, (diagnosis,)).fetchall()
            return [
                Patient(
                    patient_id=r["patient_id"],
                    external_id=r["external_id"],
                    age=r["age"],
                    sex=r["sex"],
                    diagnosis=r["diagnosis"]
                ) for r in rows
            ]

class SequenceRepository:
    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path

    def find_by_gene_pattern(self, pattern: str) -> List[Sequence]:
        """استرجاع التسلسلات التي يطابق رمز جينها نمطاً معينا (مثل BRCA%).
        Retrieve sequences whose gene symbol matches a given pattern (e.g., BRCA%).
```

```
"""
query = """
    SELECT sequence_id, sample_id, gene_symbol,
           sequence_text, length, gc_content
    FROM sequences
    WHERE gene_symbol LIKE ?
    ORDER BY gene_symbol ASC, length DESC
"""
with sqlite3.connect(self.db_path) as conn:
    conn.row_factory = sqlite3.Row
    rows = conn.execute(query, (pattern,)).fetchall()
    return [
        Sequence(
            sequence_id=r["sequence_id"],
            sample_id=r["sample_id"],
            gene_symbol=r["gene_symbol"],
            sequence_text=r["sequence_text"],
            length=r["length"],
            gc_content=r["gc_content"]
        ) for r in rows
    ]
```

تحليل هذا الكود:

Analysis of this Code:

1. كائنات OOP: نستخدم @dataclass لتحديد بنية الكائنات التي سيعيدها المستودع، مما يحافظ على تماسك البيانات وسهولة التعامل معها في باقي التطبيق.

2. معلمات مرنة: دوال مثل find_by_age_range و find_by_diagnosis و find_by_gene_pattern تقبل معاملات ديناميكية، مما يجعل الكود قابلاً لإعادة الاستخدام للعديد من الأسئلة.

3. التحويل التلقائي: نقوم بتحويل كل صف (Row) من SQLite إلى كائن OOP مناسب باستخدام الفهم (List Comprehension) و row_factory.

5. OOP Objects: We use @dataclass to define the structure of objects the repository will return, maintaining data cohesion and ease of handling in the rest of the application.

6. Flexible Parameters: Functions like find_by_age_range, find_by_diagnosis, and find_by_gene_pattern accept dynamic parameters, making the code reusable for many questions.

4. فصل المسؤوليات: المستودع (Repository) مسؤول عن الاستعلام وتحويل النتائج فقط؛ ولا يحتوي على منطق عمل (Business Logic).

7. Automatic Conversion: We convert each Row from SQLite to an appropriate OOP object using List Comprehension and row_factory.

8. Separation of Concerns: The Repository is responsible only for querying and converting results; it contains no Business Logic.

الفصل التاسع: GROUP BY / HAVING / Aggregation — تحويل البيانات إلى معرفة

Chapter Nine: GROUP BY / HAVING / Aggregation — Transforming Data into Knowledge

مختبر: تجميع القياسات المكررة

Laboratory: Aggregating Replicated Measurements

تحوّل التجميعات الملاحظات على مستوى الصف إلى ملخص علمي، لكن مفتاح التجميع هو الذي يحدد المجتمع المدروس. اجمع بحسب وحدات بيولوجية ذات معنى: عينة أو مريض أو جين أو تجربة أو كائن.

Aggregation turns row-level observations into a scientific summary, but the grouping key defines the population. Group by biological units that have a legitimate interpretation: sample, patient, gene, experiment, or organism.

يعمل HAVING على تصفية المجموعات بعد التجميع. وهو ليس بديلًا عن WHERE لأن WHERE تصفي الصفوف قبل تكوين المجموعات.

HAVING filters groups after aggregation. It is not a substitute for WHERE, because WHERE filters individual rows before groups exist.

ملخص بيولوجي على مستوى المجموعة

Group-level biological summary

```sql
SELECT gene_symbol, COUNT(DISTINCT sample_id) AS samples, AVG(expression_value) AS mean_expression
FROM expression_values
WHERE assay_version = ?
GROUP BY gene_symbol
HAVING COUNT(DISTINCT sample_id) >= ?
ORDER BY mean_expression DESC;
```

SQL

الشكل 9 — الخريطة المفاهيمية للفصل

Figure 9 — Conceptual map of the chapter

الشكل 9 — سير العمل الهندسي للفصل

Figure 9 — Engineering workflow of the chapter

هذه الخطوة لا تبدأ من الصفر؛ بل تعتمد على ما سبق بناؤه، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

From the perspective of the book's complete progression, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

9.5 GROUP BY داخل خدمة إحصائية

9.5 GROUP BY Inside a Statistical Service.

عندما يصبح الاستعلام تجميعًا علميًا، لا تجعل SQL مجرد نص مجهول داخل البرنامج. امنح النتيجة كائنًا واضحًا يصف ما تم حسابه.

When a query becomes a scientific aggregation, do not leave SQL as an anonymous string inside the program. Give its result a clear object that describes what was computed.

Python

```
from dataclasses import dataclass
import sqlite3


@dataclass
class GeneCount:
    gene_symbol: str
    sample_count: int


class ExpressionRepository:
    def __init__(self, db_path: str):
```

```
        self.db_path = db_path

    def sample_counts(self) -> list[GeneCount]:
        sql = """
        SELECT g.symbol, COUNT(DISTINCT e.sample_id)
        FROM genes g
        JOIN expression e ON e.gene_id = g.gene_id
        GROUP BY g.gene_id, g.symbol
        """
        with sqlite3.connect(self.db_path) as conn:
            rows = conn.execute(sql).fetchall()
        return [GeneCount(*row) for row in rows]
```

الاستعلامات الفردية (الصفوف المنفردة) مفيدة، لكن القوة الحقيقية لـ SQL تظهر عندما نلخص البيانات ونجمعها. دوال التجميع (Aggregate Functions) مثل MAX، MIN، AVG، COUNT، و SUM، تسمح لنا بالإجابة عن أسئلة مثل "كم عدد المرضى؟" أو "ما هو متوسط العمر؟". GROUP BY يسمح لنا بحساب هذه الإحصائيات لكل مجموعة (مثل لكل تشخيص). HAVING يسمح لنا بتصفية المجموعات بعد تجميعها.

Individual queries (single rows) are useful, but the real power of SQL emerges when we summarize and aggregate data. Aggregate Functions like COUNT, AVG, MIN, MAX, and SUM allow us to answer questions like "How many patients?" or "What is the average age?". GROUP BY allows us to calculate these statistics for each group (e.g., for each diagnosis). HAVING allows us to filter groups after aggregation.

9.1 دوال التجميع الأساسية

9.1 Basic Aggregate Functions.

لنبدأ بإحصائيات بسيطة على جدول patients:

Let's start with simple statistics on the patients table:

SQL

```
-- إجمالي عدد المرضى
-- Total number of patients
SELECT COUNT(*) AS total_patients FROM patients;


-- متوسط عمر المرضى (مع تجاهل القيم الفارغة)
```

```
-- Average age of patients (ignoring null values)
SELECT AVG(age) AS average_age FROM patients;

-- الحد الأقصى والأدنى للعمر
-- Maximum and minimum age
SELECT MAX(age) AS oldest, MIN(age) AS youngest FROM patients;

-- عدد التشخيصات المختلفة (القيم الفريدة)
-- Number of distinct diagnoses
SELECT COUNT(DISTINCT diagnosis) AS distinct_diagnoses FROM patients;
```

9.2 GROUP BY — تجميع البيانات حسب الفئات

9.2 GROUP BY — Grouping Data by Categories.

GROUP BY هو الأداة التي تحول الأرقام الأولية إلى جداول تلخيصية. مثلاً، حساب عدد المرضى لكل تشخيص.

GROUP BY is the tool that turns raw numbers into summary tables. For example, calculating the number of patients per diagnosis.

SQL

```sql
-- عدد المرضى لكل تشخيص (مرتب تنازلياً حسب العدد)
-- Patient count per diagnosis (ordered descending by count)
SELECT
    diagnosis,
    COUNT(*) AS patient_count,
    AVG(age) AS avg_age
FROM patients
WHERE diagnosis IS NOT NULL
GROUP BY diagnosis
ORDER BY patient_count DESC;
```

مثال آخر: تحليل العينات حسب النسيج:

Another example: Analyzing samples by tissue:

SQL

```sql
-- عدد العينات ومتوسط جودة RNA لكل نوع نسيج
-- Sample count and average RNA quality per tissue type
SELECT
    tissue_type,
    COUNT(*) AS sample_count,
    AVG(rna_quality) AS avg_rna_quality
FROM samples
GROUP BY tissue_type
ORDER BY sample_count DESC;
```

9.3 HAVING — تصفية المجموعات بعد التجميع

9.3 HAVING — Filtering Groups After Aggregation.

WHERE يصفّي الصفوف الفردية قبل التجميع. HAVING يصفّي المجموعات بعد التجميع. مثلاً، نريد فقط التشخيصات التي تضم أكثر من 5 مرضى.

WHERE filters individual rows before aggregation. HAVING filters groups after aggregation. For example, we only want diagnoses with more than 5 patients.

SQL

```
-- التشخيصات التي تضم أكثر من 5 مرضى، مع متوسط أعمارهم
-- Diagnoses with more than 5 patients, along with their average age
SELECT
    diagnosis,
    COUNT(*) AS patient_count,
    AVG(age) AS avg_age
FROM patients
WHERE diagnosis IS NOT NULL
GROUP BY diagnosis
HAVING COUNT(*) > 5
ORDER BY patient_count DESC;
```

مثال آخر على التسلسلات والطفرات:

Another example on sequences and mutations:

SQL

```
-- الجينات (gene_symbol) التي لديها أكثر من 3 طفرات مسجلة
-- Genes (gene_symbol) with more than 3 recorded mutations
SELECT
    seq.gene_symbol,
    COUNT(m.mutation_id) AS mutation_count
FROM sequences seq
JOIN mutations m ON seq.sequence_id = m.sequence_id
GROUP BY seq.gene_symbol
HAVING COUNT(m.mutation_id) > 3
ORDER BY mutation_count DESC;
```

(ملاحظة: هذا المثال يستخدم JOIN الذي سنشرحه بالتفصيل في الفصل القادم، لكنه يوضح قوة HAVING مع الربط بين الجداول).

(Note: This example uses JOIN, which we will explain in detail in the next chapter, but it demonstrates the power of HAVING with table joins.).

9.4 ربط GROUP BY بـ OOP (عملي)

9.4 Connecting GROUP BY to OOP (Practical).

سننشئ كائناً جديداً (DiagnosisSummary) لتمثيل نتائج التجميع، ومستودعاً (StatisticsRepository) لإجراء التحليلات الإحصائية.

We will create a new object (DiagnosisSummary) to represent aggregation results, and a repository (StatisticsRepository) to perform statistical analyses.

Python

```python
from dataclasses import dataclass
from typing import List

@dataclass
class DiagnosisSummary:
    diagnosis: str
    patient_count: int
    avg_age: float

@dataclass
class TissueSummary:
    tissue_type: str
```

```
    sample_count: int
    avg_rna_quality: float

class StatisticsRepository:
    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path

    def summarize_patients_by_diagnosis(self, min_count: int = 0) -> List[DiagnosisSummary]:
        """الحصول على ملخص للمرضى حسب التشخيص، مع شرط الحد الأدنى للعدد.
        Get a summary of patients by diagnosis, with a minimum count condition.
        """
        query = """
            SELECT
                diagnosis,
                COUNT(*) AS patient_count,
                AVG(age) AS avg_age
            FROM patients
            WHERE diagnosis IS NOT NULL
            GROUP BY diagnosis
            HAVING COUNT(*) >= ?
            ORDER BY patient_count DESC
        """
        with sqlite3.connect(self.db_path) as conn:
            conn.row_factory = sqlite3.Row
            rows = conn.execute(query, (min_count,)).fetchall()
            return [
                DiagnosisSummary(
                    diagnosis=r["diagnosis"],
                    patient_count=r["patient_count"],
                    avg_age=r["avg_age"] if r["avg_age"] is not None else 0.0
                ) for r in rows
            ]
```

```
def summarize_samples_by_tissue(self) -> List[TissueSummary]:
    """الحصول على ملخص للعينات حسب نوع النسيج.
    Get a summary of samples by tissue type.
    """
    query = """
        SELECT
            tissue_type,
            COUNT(*) AS sample_count,
            AVG(rna_quality) AS avg_rna_quality
        FROM samples
        WHERE tissue_type IS NOT NULL
        GROUP BY tissue_type
        ORDER BY sample_count DESC
    """
    with sqlite3.connect(self.db_path) as conn:
        conn.row_factory = sqlite3.Row
        rows = conn.execute(query).fetchall()
        return [
            TissueSummary(
                tissue_type=r["tissue_type"],
                sample_count=r["sample_count"],
                avg_rna_quality=r["avg_rna_quality"] if r["avg_rna_quality"] is not None else 0.0
            ) for r in rows
        ]
```

الفصل العاشر: JOIN — نبض القلب العلاقي

Chapter Ten: JOIN — The Relational Heartbeat

مختبر: استعلامات الأدلة many-to-many

Laboratory: Many-to-Many Evidence Queries

يجب أن يتبع JOIN علاقة معلنة. تنتمي العلاقة many-to-many إلى جدول ربط بدل نص مكرر، لأن العلاقة نفسها قد تحمل دليلًا وثقة وتواريخ.

A JOIN should follow a declared relationship. A many-to-many association belongs in a junction table rather than in repeated text, because the association itself may carry evidence, confidence, and dates.

عندما يحتوي الاستعلام على عدة عمليات JOIN، تحقق من الحفاظ على الكاردينالية. فقد يؤدي توسع one-to-many غير المقصود إلى تضخيم الأعداد بصمت.

When several joins are involved, verify that the query preserves cardinality. An accidental one-to-many expansion can silently inflate counts.

استعلام متعدد الجداول واعٍ بالدليل

Evidence-aware multi-table query

```sql
SELECT m.hgvs, d.drug_name, e.evidence_level, e.confidence
FROM mutations AS m
JOIN mutation_drug AS md ON md.mutation_id = m.mutation_id
JOIN drugs AS d ON d.drug_id = md.drug_id
JOIN resistance_evidence AS e ON e.mutation_drug_id = md.mutation_drug_id
WHERE m.gene_symbol = ?
ORDER BY e.confidence DESC, d.drug_name;
```

SQL

الشكل 10 — الخريطة المفاهيمية للفصل

Figure 10 — Conceptual map of the chapter

الشكل 10 — سير العمل الهندسي للفصل

Figure 10 — Engineering workflow of the chapter

في الموضع الحالي من الرحلة التعليمية، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

This step does not start from scratch; it builds on what came before, you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

10.5 JOIN داخل Repository — تحويل العلاقات إلى كائنات

10.5 JOIN Inside a Repository — Turning Relationships into Objects.

JOIN هو المكان الذي تتحول فيه خريطة العلاقات إلى نتيجة عملية. اجعل الكائن الناتج يمثل السؤال العلمي بدل أن يترك بقية النظام يتعامل مع tuple غير موصوف.

JOIN is where the relationship map becomes an operational result. Let the resulting object represent the scientific question instead of forcing the rest of the system to manipulate an unexplained tuple.

Python

```
from dataclasses import dataclass
import sqlite3


@dataclass
class MutationResistance:
    gene_symbol: str
    mutation: str
    phenotype: str


def resistant_mutations(db_path: str) -> list[MutationResistance]:
    sql = """
    SELECT g.symbol, m.change, r.phenotype
    FROM genes g
    JOIN mutations m ON m.gene_id = g.gene_id
    JOIN resistance_observations r
      ON r.mutation_id = m.mutation_id
    """
    with sqlite3.connect(db_path) as conn:
```

```
    rows = conn.execute(sql).fetchall()
return [MutationResistance(*row) for row in rows]
```

هذا الفصل هو الفصل المحوري في الباب الثالث، بل في الجزء الثالث بأكمله. JOIN هو ما يميز قواعد البيانات العلائقية عن أي شيء آخر. هو الأداة التي تجمع الجداول المنفصلة معاً لتشكل الصورة الكاملة للبيانات. بدون JOIN، تكون الجداول مجرد جزر معزولة. مع JOIN، تصبح شبكة مترابطة.

This chapter is the pivotal chapter of Unit Three, indeed of the entire Part Three. JOIN is what distinguishes relational databases from everything else. It is the tool that brings separate tables together to form the complete picture of the data. Without JOIN, tables are isolated islands. With JOIN, they become an interconnected network.

السؤال الذي سيصرخ به القارئ: "آه... هذه هي الطريقة التي أربط بها Sample مع Experiment مع Sequence!"

The question the reader will exclaim: "Ah... this is how I link Sample with Experiment with Sequence!"

هناك عدة أنواع من JOIN، لكن النوع الأكثر استخداماً والأكثر أهمية هو INNER JOIN، والذي يعيد فقط الصفوف التي لها تطابق في كلا الجدولين. سنستخدمه بشكل مكثف، ثم نقدم LEFT JOIN للحالات التي قد لا يكون فيها تطابق (مثل عينة بدون تسلسلات).

There are several types of JOIN, but the most used and most important is INNER JOIN, which returns only rows that have a match in both tables. We will use it extensively, then introduce LEFT JOIN for cases where a match might not exist (e.g., a sample without sequences).

10.1 INNER JOIN — الربط الداخلي (التطابق التام)

10.1 INNER JOIN — Internal Linking (Exact Match).

المثال الكلاسيكي: ربط العينات بالمرضى.

The Classic Example: Linking Samples to Patients.

لنفترض أننا نريد عرض العينات مع بيانات المرضى (العمر، التشخيص) بدلاً من مجرد patient_id الرقمي.

Suppose we want to display samples with patient data (age, diagnosis) instead of just the numeric patient_id.

SQL

```
-- جلب العينات مع بيانات المرضى المرتبطين بها --
-- Fetch samples with their associated patient data
SELECT
    s.sample_id,
    s.tissue_type,
    s.collection_date,
    p.external_id AS patient_external_id,
    p.age AS patient_age,
    p.diagnosis
FROM samples AS s
INNER JOIN patients AS p ON s.patient_id = p.patient_id
ORDER BY p.age DESC;
```

The Golden Example (Three Tables): Linking Sample ← Sequence ← Mutation.

المثال الذهبي (ثلاثة جداول): ربط العينة ← التسلسل ← الطفرة.

السؤال: "أعرض جميع الطفرات مع العينة التي تنتمي إليها ورمز الجين."

Question: "Show all mutations with the sample they belong to and the gene symbol."

SQL

```
SELECT
    m.mutation_id,
    m.position,
    m.ref_base,
    m.alt_base,
    seq.gene_symbol,
    seq.sequence_id,
    s.sample_id,
    s.tissue_type
FROM mutations m
INNER JOIN sequences seq ON m.sequence_id = seq.sequence_id
INNER JOIN samples s ON seq.sample_id = s.sample_id
ORDER BY seq.gene_symbol, m.position;
```

The Comprehensive Scientific Example (Four Tables): Mutations ← Resistance (via Junction Table).

المثال العلمي الشامل (أربعة جداول): الطفرات ← المقاومة (عبر الجدول الوسيط).

السؤال العلمي الكامل: "أعرض جميع الطفرات، والأدوية المرتبطة بها، ودرجة الثقة، ورمز الجين، ونوع النسيج."

The complete scientific question: "Show all mutations, the drugs they are associated with, the confidence score, the gene symbol, and the tissue type."

SQL

```
SELECT
    m.position,
    m.ref_base,
    m.alt_base,
    m.effect,
    seq.gene_symbol,
    r.drug_name,
    mr.confidence,
    s.tissue_type
FROM mutations m
INNER JOIN sequences seq ON m.sequence_id = seq.sequence_id
```

```sql
INNER JOIN samples s ON seq.sample_id = s.sample_id
INNER JOIN mutation_resistance mr ON m.mutation_id = mr.mutation_id
INNER JOIN resistance r ON mr.resistance_id = r.resistance_id
WHERE mr.confidence > 0.7
ORDER BY seq.gene_symbol, m.position, r.drug_name;
```

هذا الاستعلام هو الترجمة المباشرة للسؤال: "أعطني جميع الطفرات ذات الصلة بالمقاومة (بثقة > 0.7)، مع الجين والنسيج والدواء." لاحظ كيف أن الجدول الوسيط mutation_resistance يعمل كجسر لربط mutations بـ resistance في علاقة M:N. هذا هو بالضبط ما صممناه في الباب الثاني.

This query is the direct translation of the question: "Give me all resistance-related mutations (confidence > 0.7), along with the gene, tissue, and drug." Note how the junction table mutation_resistance acts as a bridge to link mutations with resistance in an M:N relationship. This is exactly what we designed in Unit Two.

10.2 LEFT JOIN —— الربط الخارجي (الحفاظ على الصفوف غير المتطابقة)

10.2 LEFT JOIN — External Linking (Preserving Non-Matching Rows).

ماذا لو أردنا عرض جميع العينات، حتى تلك التي ليس لها تسلسلات مسجلة بعد؟ INNER JOIN سيتجاهلها. هنا يأتي LEFT JOIN (أو LEFT OUTER JOIN) الذي يحافظ على جميع صفوف الجدول الأيسر، ويضع NULL للصفوف غير المتطابقة من الجدول الأيمن.

What if we want to display all samples, even those that do not have sequences recorded yet? INNER JOIN would ignore them. This is where LEFT JOIN (or LEFT OUTER JOIN) comes in, preserving all rows from the left table and placing NULL for non-matching rows from the right table.

SQL

```
-- عرض جميع العينات، مع بيانات التسلسل إذا كانت موجودة
-- Display all samples, with sequence data if available
SELECT
    s.sample_id,
    s.tissue_type,
    seq.sequence_id,
    seq.gene_symbol,
    seq.length
FROM samples s
LEFT JOIN sequences seq ON s.sample_id = seq.sample_id
ORDER BY s.sample_id;
```

هذا مفيد بشكل خاص في مراحل إدخال البيانات، لمعرفة العينات التي لم تُعالج بعد.

This is especially useful during data entry stages, to identify samples that have not yet been processed.

10.3 ربط JOIN بـ OOP (عملي — استعلام علمي معقد)

10.3 Connecting JOIN to OOP (Practical — Complex Scientific Query).

لنقم بإنشاء كائن OOP لتمثيل النتيجة العلمية الكاملة التي تجمع بيانات من أربعة جداول. هذا الكائن يعكس البنية المنطقية التي يحتاجها عالم الأحياء، وليس مجرد صفوف قاعدة بيانات.

Let's create an OOP object to represent the complete scientific result combining data from four tables. This object reflects the logical structure needed by a biologist, not just database rows.

Python

```
from dataclasses import dataclass
from typing import List, Optional


@dataclass
class MutationResistanceView:
    """عرض علمي يجمع بيانات الطفرة، الجين، النسيج، والدواء مع درجة الثقة.
    Scientific view combining mutation, gene, tissue, and drug data with confidence.
    """
    position: int
    ref_base: str
    alt_base: str
    effect: Optional[str]
    gene_symbol: str
```

```
    tissue_type: str
    drug_name: str
    confidence: float

class ScientificQueryRepository:
    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path

    def find_resistance_mutations(self, min_confidence: float = 0.7) ->
List[MutationResistanceView]:
        """استرجاع الطفرات المرتبطة بالمقاومة بثقة أعلى من حد معين.
        Retrieve mutations associated with resistance with confidence above a threshold.
        """
        query = """
            SELECT
                m.position,
                m.ref_base,
                m.alt_base,
                m.effect,
                seq.gene_symbol,
                s.tissue_type,
                r.drug_name,
                mr.confidence
            FROM mutations m
            INNER JOIN sequences seq ON m.sequence_id = seq.sequence_id
            INNER JOIN samples s ON seq.sample_id = s.sample_id
            INNER JOIN mutation_resistance mr ON m.mutation_id = mr.mutation_id
            INNER JOIN resistance r ON mr.resistance_id = r.resistance_id
            WHERE mr.confidence >= ?
            ORDER BY seq.gene_symbol, m.position, r.drug_name
        """
        with sqlite3.connect(self.db_path) as conn:
```

```python
        conn.row_factory = sqlite3.Row
        rows = conn.execute(query, (min_confidence,)).fetchall()
        return [
            MutationResistanceView(
                position=r["position"],
                ref_base=r["ref_base"],
                alt_base=r["alt_base"],
                effect=r["effect"],
                gene_symbol=r["gene_symbol"],
                tissue_type=r["tissue_type"],
                drug_name=r["drug_name"],
                confidence=r["confidence"]
            ) for r in rows
        ]

    def find_patients_with_resistance_gene(self, gene_symbol: str) -> List[str]:
        """استرجاع المعرفات الخارجية للمرضى الذين لديهم طفرة في جين معين مرتبط بمقاومة.
        Retrieve external IDs of patients who have a mutation in a specific gene linked to resistance.
        """
        query = """
            SELECT DISTINCT p.external_id
            FROM patients p
            INNER JOIN samples s ON p.patient_id = s.patient_id
            INNER JOIN sequences seq ON s.sample_id = seq.sample_id
            INNER JOIN mutations m ON seq.sequence_id = m.sequence_id
            INNER JOIN mutation_resistance mr ON m.mutation_id = mr.mutation_id
            WHERE seq.gene_symbol = ?
        """
        with sqlite3.connect(self.db_path) as conn:
            rows = conn.execute(query, (gene_symbol,)).fetchall()
            return [row[0] for row in rows]
```

تحليل هذا الكود:

Analysis of this Code:

1. كائنات ذات معنى علمي: MutationResistanceView ليس مجرد صف SQL، بل هو تمثيل مباشر للإجابة عن سؤال بيولوجي: طفرة تؤدي إلى مقاومة دواء معين.
2. استعلامات ذات أربع JOINs: الكود يعكس بنية المخطط تماماً: patients ← samples ← sequences ← mutations (في الدالة الثانية) وربط mutation_resistance مع resistance.

4. Scientifically Meaningful Objects: MutationResistanceView is not just an SQL row; it is a direct representation of an answer to a biological question: a mutation leading to resistance to a specific drug.

5. Queries with Four JOINS: The code perfectly reflects the Schema structure: mutations ← sequences ← samples ← patients (in the second function) and linking mutation_resistance with resistance.

3. إعادة الاستخدام: يمكن تمديد هذا المستودع بأي استعلام معقد دون تكرار منطق الاتصال بقاعدة البيانات.

6. Reusability: This repository can be extended with any complex query without repeating database connection logic.

الفصل الحادي عشر: **Subqueries** و **CTEs** — تقسيم الأسئلة المعقدة

Chapter Eleven: Subqueries and CTEs — Breaking Down Complex Questions

مختبر: تفكيك السؤال العلمي إلى علاقات

Laboratory: Breaking a Scientific Question into Relations

الـCTE علاقة وسيطة تحمل اسمًا. ابنِ الاستعلام على مراحل حتى يمكن فحص كل قرار علمي بصورة مستقلة.

A CTE is a named intermediate relation. Build the query in stages so each scientific decision can be inspected independently.

يكون CTE المقروء مفيدًا للتحقق حتى عندما يستطيع مخطط التنفيذ تحسين صياغة مكافئة بطريقة مختلفة.

A readable CTE is useful for verification even when the query planner can optimize an equivalent formulation differently.

تصفية علمية على مراحل

Staged scientific filtering

```
WITH eligible AS (
   SELECT sample_id
   FROM samples
   WHERE quality_score >= ?
), qualifying AS (
   SELECT m.mutation_id, m.sample_id
   FROM mutations AS m
   JOIN eligible AS e ON e.sample_id = m.sample_id
   WHERE m.pathogenicity_score >= ?
)
SELECT COUNT(*) AS qualifying_mutations
FROM qualifying;
```

SQL

الشكل 11 — الخريطة المفاهيمية للفصل

Figure 11 — Conceptual map of the chapter

بعد العمل السابق على الكائنات والبيانات، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

الشكل 11 — سير العمل الهندسي للفصل

Figure 11 — Engineering workflow of the chapter

11.5 CTE مع Service Layer — تفكيك السؤال المعقد

At this point in the learning path, you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

عندما يصبح السؤال متعدد المراحل، استخدم CTE لتسمية المراحل المنطقية في SQL، ثم استخدم طبقة خدمة في Python لتفسير النتيجة ضمن سياق المشروع.

11.5 CTE with a Service Layer — Decomposing a Complex Question.

When a question becomes multi-stage, use CTEs to name logical stages in SQL, then use a Python service layer to interpret the result within the project context.

Python

```
from dataclasses import dataclass
import sqlite3

@dataclass
class ResistanceSummary:
    gene_symbol: str
    resistant_samples: int

class ResistanceService:
    def __init__(self, db_path: str):
        self.db_path = db_path
```

```
def summarize(self) -> list[ResistanceSummary]:
    sql = """
    WITH resistant AS (
        SELECT mutation_id, COUNT(DISTINCT sample_id) AS n
        FROM resistance_observations
        WHERE phenotype = 'resistant'
        GROUP BY mutation_id
    )
    SELECT g.symbol, SUM(resistant.n)
    FROM resistant
    JOIN mutations m ON m.mutation_id = resistant.mutation_id
    JOIN genes g ON g.gene_id = m.gene_id
    GROUP BY g.gene_id, g.symbol
    """
    with sqlite3.connect(self.db_path) as conn:
        rows = conn.execute(sql).fetchall()
    return [ResistanceSummary(*row) for row in rows]
```

بعض الأسئلة تحتاج إلى أكثر من استعلام بسيط أو JOIN. قد تحتاج إلى نتيجة استعلام كشرط لاستعلام آخر (Subquery)، أو قد ترغب في تبسيط استعلام معقد بإعطائه اسماً مؤقتاً (CTE — Common Table Expression). هذا يجعل SQL قابلة للقراءة والصيانة مثل كود Python المنظم.

Some questions require more than a simple query or JOIN. You might need the result of one query as a condition for another (Subquery), or you might want to simplify a complex query by giving it a temporary name (CTE — Common Table Expression). This makes SQL as readable and maintainable as well-structured Python code.

11.1 Subquery في WHERE — التصفية بناءً على استعلام آخر

11.1 Subquery in WHERE — Filtering Based on Another Query.

السؤال: "أعطني المرضى الذين لديهم عينات ذات جودة RNA أعلى من 8."

Question: "Give me patients who have samples with RNA quality higher than 8."

بدلاً من استخدام JOIN ثم DISTINCT، نستخدم Subquery لجلب patient_id من العينات ذات الجودة العالية أولاً، ثم استعلام خارجي لجلب بيانات المرضى.

Instead of using JOIN then DISTINCT, we use a Subquery to fetch patient_id from high-quality samples first, then an outer query to fetch patient data.

SQL

```
SELECT external_id, age, diagnosis
FROM patients
WHERE patient_id IN (
   SELECT DISTINCT patient_id
   FROM samples
   WHERE rna_quality > 8
);
```

السؤال: "أعطني الجينات التي يتجاوز متوسط طولها في العينات المتوفرة 2000 قاعدة."

Question: "Give me genes whose average length across available samples exceeds 2000 bases."

SQL

```
SELECT gene_symbol, AVG(length) AS avg_length
FROM sequences
GROUP BY gene_symbol
HAVING AVG(length) > (
    SELECT AVG(length) FROM sequences
);
```

هنا، القيمة المقارنة في HAVING هي نتيجة Subquery واحدة (المتوسط العام للطول).

Here, the comparison value in HAVING is the result of a single Subquery (the overall average length).

11.2 Subquery في FROM — الاستعلام كجدول مؤقت

11.2 Subquery in FROM — Query as a Temporary Table.

يمكننا استخدام نتيجة استعلام كجدول مؤقت في FROM. هذا مفيد جداً لتجميع البيانات ثم الانضمام إليها كوحدة واحدة.

We can use the result of a query as a temporary table in FROM. This is very useful for aggregating data and then joining it as a single unit.

السؤال: "لكل تشخيص، أظهر عدد المرضى ومتوسط أعمارهم، ولكن فقط إذا كان متوسط العمر أكبر من متوسط العمر الكلي للجميع."

Question: "For each diagnosis, show the patient count and average age, but only if the average age is greater than the overall average age of everyone."

SQL

```
SELECT
   diagnosis_summary.diagnosis,
   diagnosis_summary.patient_count,
   diagnosis_summary.avg_age
FROM (
   SELECT
      diagnosis,
      COUNT(*) AS patient_count,
      AVG(age) AS avg_age
   FROM patients
   WHERE diagnosis IS NOT NULL
   GROUP BY diagnosis
) AS diagnosis_summary
```

```
WHERE diagnosis_summary.avg_age > (
    SELECT AVG(age) FROM patients
);
```

11.3 CTEs (WITH) — كتابة منظمة وقابلة للقراءة

11.3 CTEs (WITH) — Structured and Readable Writing.

CTEs (Common Table Expressions) هي تطور طبيعي للـ Subquery. بدلاً من كتابة Subquery متداخلة قد تصبح مربكة، نستخدم WITH لتعريف استعلام مؤقت باسم واضح، ثم نستخدمه في الاستعلام الرئيسي. هذا يشبه تعريف دالة مساعدة في Python قبل استخدامها.

CTEs (Common Table Expressions) are a natural evolution of Subqueries. Instead of writing a nested Subquery that might become confusing, we use WITH to define a temporary query with a clear name, then use it in the main query. This is like defining a helper function in Python before using it.

نفس السؤال السابق، لكن باستخدام CTE:

The same question as before, but using a CTE:

SQL

```
WITH diagnosis_summary AS (
    SELECT
        diagnosis,
        COUNT(*) AS patient_count,
        AVG(age) AS avg_age
    FROM patients
    WHERE diagnosis IS NOT NULL
    GROUP BY diagnosis
),
overall_avg AS (
    SELECT AVG(age) AS avg_age FROM patients
)
SELECT
    diagnosis_summary.diagnosis,
    diagnosis_summary.patient_count,
    diagnosis_summary.avg_age
FROM diagnosis_summary
WHERE diagnosis_summary.avg_age > (SELECT avg_age FROM overall_avg)
ORDER BY diagnosis_summary.patient_count DESC;
```

مثال علمي معقد باستخدام CTE: تحليل الجينات الأكثر طفرة.

Complex Scientific Example with CTE: Analyzing the Most Mutated Genes.

السؤال: "أعرض الجينات التي لديها أكثر من 3 طفرات، مع حساب متوسط درجة الثقة للطفرات المرتبطة بمقاومة الأدوية."

Question: "Show genes with more than 3 mutations, calculating the average confidence score for drug-resistance-associated mutations."

SQL

```
WITH gene_mutation_count AS (
    SELECT
        seq.gene_symbol,
        COUNT(m.mutation_id) AS mutation_count
    FROM sequences seq
    LEFT JOIN mutations m ON seq.sequence_id = m.sequence_id
    GROUP BY seq.gene_symbol
),
gene_resistance_confidence AS (
    SELECT
        seq.gene_symbol,
        AVG(mr.confidence) AS avg_confidence
    FROM sequences seq
    LEFT JOIN mutations m ON seq.sequence_id = m.sequence_id
    LEFT JOIN mutation_resistance mr ON m.mutation_id = mr.mutation_id
    GROUP BY seq.gene_symbol
)
```

```sql
SELECT
    gmc.gene_symbol,
    gmc.mutation_count,
    grc.avg_confidence
FROM gene_mutation_count gmc
LEFT JOIN gene_resistance_confidence grc ON gmc.gene_symbol = grc.gene_symbol
WHERE gmc.mutation_count > 3
ORDER BY gmc.mutation_count DESC;
```

هذا الاستعلام واضح ومنظم: نحدد الجينات وعدد طفراتها في CTE أولاً، ومتوسط درجة الثقة في CTE ثان، ثم نجمعهما في الاستعلام النهائي. هذا أقرب إلى البرمجة المنظمة (Structured Programming) منه إلى SQL التقليدية.

This query is clear and structured: we define genes and their mutation count in the first CTE, and the average confidence score in the second CTE, then combine them in the final query. This is closer to Structured Programming than traditional SQL.

11.4 ربط Subqueries و CTEs بـ OOP

11.4 Connecting Subqueries and CTEs to OOP.

Python

```
class AdvancedQueryRepository:
    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path

    def get_high_mutation_genes(self, min_mutations: int = 3) -> List[dict]:
        """استرجاع الجينات التي يتجاوز عدد طفراتها حداً معيناً، مع متوسط الثقة.
        Retrieve genes with mutation count exceeding a threshold, with average confidence.
        """
        query = """
        WITH gene_mutation_count AS (
            SELECT
                seq.gene_symbol,
                COUNT(m.mutation_id) AS mutation_count
            FROM sequences seq
            LEFT JOIN mutations m ON seq.sequence_id = m.sequence_id
            GROUP BY seq.gene_symbol
        ),
        gene_resistance_confidence AS (
            SELECT
                seq.gene_symbol,
                AVG(mr.confidence) AS avg_confidence
            FROM sequences seq
            LEFT JOIN mutations m ON seq.sequence_id = m.sequence_id
            LEFT JOIN mutation_resistance mr ON m.mutation_id = mr.mutation_id
            GROUP BY seq.gene_symbol
        )
```

```
SELECT
    gmc.gene_symbol,
    gmc.mutation_count,
    grc.avg_confidence
FROM gene_mutation_count gmc
LEFT JOIN gene_resistance_confidence grc ON gmc.gene_symbol = grc.gene_symbol
WHERE gmc.mutation_count >= ?
ORDER BY gmc.mutation_count DESC
"""
with sqlite3.connect(self.db_path) as conn:
    conn.row_factory = sqlite3.Row
    rows = conn.execute(query, (min_mutations,)).fetchall()
    return [dict(row) for row in rows]  # تحويل إلى قاموس لتسهيل العرض
```

الفصل الثاني عشر: **SQL** كمحرك سؤال علمي — العودة إلى المنهجية

Chapter Twelve: SQL as a Scientific Question Engine — Returning to the Methodology

مختبر: جعل الاستعلام قابلًا للتدقيق الذاتي

Laboratory: Making the Query Self-Auditing

يجب أن يجعل الاستعلام العلمي المجتمع المدروس والمقياس والإصدار والترتيب ظاهرة بوضوح. تجنب الترتيب الضمني ولا تخلط منطق العرض بالاختيار العلمي.

A scientific query should make its cohort, metric, version, and ordering visible. Avoid implicit ordering and avoid mixing display logic with scientific selection.

يمكن لخطة التنفيذ أن تكشف ما إذا كان الفهرس أو ترتيب الربط أو المسح مسؤولًا عن الكلفة المشاهدة. استخدم الخطة قبل إطلاق أحكام حول الأداء.

A query plan can reveal whether an index, join order, or scan is responsible for the observed cost. Use the plan before making performance claims.

فحص خطة تنفيذ الاستعلام

Inspecting a query plan

```
EXPLAIN QUERY PLAN
SELECT s.sample_id, r.resistance_score
FROM samples AS s
JOIN resistance_results AS r ON r.sample_id = s.sample_id
WHERE s.organism = ?
  AND r.resistance_score >= ?;
```

SQL

الشكل 12 — الخريطة المفاهيمية للفصل

Figure 12 — Conceptual map of the chapter

الشكل 12 — سير العمل الهندسي للفصل

Figure 12 — Engineering workflow of the chapter

في الجزء الثاني من الكتاب، وضعنا سلماً منهجياً للتحليل العلمي:

السؤال العلمي → تمثيل البيانات → الحساب → التفسير → التحقق.

الآن، نرى كيف تتلاءم SQL مع هذه السلسلة. SQL ليست مجرد أداة لاسترجاع البيانات؛ هي محرك التمثيل والحساب في هذه السلسلة عندما تكون البيانات مهيكلة وعلائقية.

In Part Two of the book, we established a methodological ladder for scientific analysis:

Scientific Question → Data Representation → Computation → Interpretation → Validation.

Now, we see how SQL fits into this chain. SQL is not just a data retrieval tool; it is the Representation and Computation engine in this chain when dealing with structured, relational data.

12.1 السؤال العلمي → تمثيل البيانات

12.1 Scientific Question → Data Representation.

لنأخذ سؤالاً علمياً حقيقياً (مبسطاً للتعليم): "هل هناك ارتباط بين عمر المريض ووجود طفرات في جين مقاومة الميثيسيلين (mecA)؟"

Let's take a real scientific question (simplified for education): "Is there a correlation between patient age and the presence of mutations in the methicillin resistance gene (mecA)?"

تمثيل البيانات في SQL هو المخطط الذي صممناه: جداول sequences، samples، patients، و mutations. السؤال يُترجم إلى استعلام يجمع هذه الجداول، ويصفى حسب gene_symbol = 'mecA'، ويعيد الأعمار مع وجود/غياب الطفرات.

The data representation in SQL is the Schema we designed: tables patients, samples, sequences, and mutations. The question translates into a query that joins these tables, filters by gene_symbol = 'mecA', and returns ages with the presence/absence of mutations.

SQL

```sql
SELECT
    p.age,
    CASE WHEN m.mutation_id IS NOT NULL THEN 1 ELSE 0 END AS has_mutation
FROM patients p
JOIN samples s ON p.patient_id = s.patient_id
JOIN sequences seq ON s.sample_id = seq.sample_id
LEFT JOIN mutations m ON seq.sequence_id = m.sequence_id
WHERE seq.gene_symbol = 'mecA';
```

12.2 الحساب ← التحليل الإحصائي الأولي

12.2 Computation → Preliminary Statistical Analysis.

بمجرد الحصول على البيانات، يمكننا إجراء حسابات إحصائية أولية مباشرة في SQL (مثل متوسط عمر المرضى الذين لديهم طفرة مقابل الذين لا يملكونها).

Once the data is retrieved, we can perform preliminary statistical calculations directly in SQL (e.g., the average age of patients with mutations versus those without).

SQL

```
SELECT
    CASE WHEN m.mutation_id IS NOT NULL THEN 'With Mutation' ELSE 'No Mutation' END AS
mutation_status,
    COUNT(DISTINCT p.patient_id) AS patient_count,
    AVG(p.age) AS avg_age,
    MIN(p.age) AS min_age,
    MAX(p.age) AS max_age
FROM patients p
JOIN samples s ON p.patient_id = s.patient_id
```

```
JOIN sequences seq ON s.sample_id = seq.sample_id
LEFT JOIN mutations m ON seq.sequence_id = m.sequence_id
WHERE seq.gene_symbol = 'mecA'
GROUP BY mutation_status;
```

هذا الاستعلام يعطي عالم الأحياء أول لمحة إحصائية عن البيانات، تماماً كما كان يفعل pandas في Python، لكن دون تحميل البيانات إلى الذاكرة المحلية.

This query gives the biologist the first statistical glimpse of the data, just as pandas would in Python, but without loading the data into local memory.

12.3 التفسير ← ربط النتيجة بالسياق البيولوجي

12.3 Interpretation → Linking Results to Biological Context.

النتائج التي يولدها SQL هي أرقام وحقائق. دور الباحث هو تفسيرها في السياق البيولوجي. مثلاً، إذا كان متوسط عمر

المرضى الذين لديهم طفرة mecA أعلى، فقد يشير ذلك إلى أن المقاومة تتراكم مع تقدم العمر (ربما بسبب التعرض المتكرر للمضادات الحيوية). لكن SQL لا تقدم هذا التفسير؛ إنها تقدم الأدلة التي تسمح به.

The results generated by SQL are numbers and facts. The researcher's role is to interpret them in the biological context. For example, if the average age of patients with the mecA mutation is higher, it might indicate that resistance accumulates with age (perhaps due to repeated antibiotic exposure). However, SQL does not offer this interpretation; it offers the evidence that allows it.

12.4 التحقق ← قابلية إعادة الإنتاج

12.4 Validation → Reproducibility.

السؤال العلمي، والاستعلام المستخدم، والمخطط، وإصدار قاعدة البيانات، كلها جزء من سجل إعادة الإنتاج. يمكن لأي باحث آخر، باستخدام نفس المخطط ونفس الاستعلام، الحصول على نفس النتائج من نفس البيانات. هذا هو جوهر التحقق العلمي في العصر الرقمي.

The scientific question, the query used, the Schema, and the database version are all part of the reproducibility record. Any other researcher, using the same Schema and the same query, can obtain the same results from the same data. This is the essence of scientific validation in the digital age.

12.5 ربط SQL بالمنهجية ككل

12.5 Connecting SQL to the Overall Methodology.

المرحلة (Stage) في الجزء الثاني (In Part Two) في الجزء الثالث (In Part Three)

السؤال العلمي ما الفرق في التعبير بين المجموعتين؟ ما ارتباط عمر المريض بطفرات mecA؟

تمثيل البيانات كائنات DataFrames، SeqRecord. مخطط (Schema) في SQLite.

الحساب دوال Python (t-test، إلخ). استعلامات SQL مع JOIN و GROUP BY.

التفسير تحليل حجم التأثير والسياق البيولوجي. تحليل المتوسطات والارتباطات الظاهرية.

التحقق إعادة تشغيل الكود على نفس البيانات. إعادة تشغيل الاستعلام على نفس قاعدة البيانات.

هذا الجدول يوضح أن SQL ليست دخيلة على المنهجية؛ إنها تطبيق مباشر لها على البيانات المهيكلة واسعة النطاق.

This table shows that SQL is not foreign to the methodology; it is a direct application of it to structured, large-scale data.

الانتقال إلى الباب الرابع:

Transition to Unit Four:

الآن وقد أتقنا الاستعلامات (SELECT، WHERE، JOIN، GROUP BY، CTEs)، وأدركنا كيف تخدم المنهج العلمي، أصبحنا جاهزين للانتقال إلى المستوى التالي: تصميم قواعد البيانات البيولوجية نفسها. في الباب الرابع، سنطبق كل ما تعلمناه لتصميم قواعد بيانات متخصصة للجينات، والبروتينات، والبيانات الوصفية للتجارب، مع التوسع في مفاهيم التطبيع

(Normalization) و إزالة التطبيع (Denormalization) حسب الحاجة. سننشئ مخططاً متكاملاً لمشروعنا العملي، ونجهز القاعدة التي ستبني عليها باقي الأبواب (Workflow ،Python Layer، والواجهة).

Now that we have mastered queries (SELECT, WHERE, JOIN, GROUP BY, CTEs), and realized how they serve the scientific methodology, we are ready to move to the next level: Designing Biological Databases themselves. In Unit Four, we will apply everything we have learned to design specialized databases for genes, proteins, and experimental metadata, expanding on Normalization and Denormalization concepts as needed. We will create an integrated Schema for our practical project, preparing the foundation upon which the rest of the units (Python Layer, Workflow, and Interface) will be built.

الباب الرابع: تصميم قواعد البيانات البيولوجية

Unit Four: Designing Biological Databases

الغرض من هذا الباب:

الانتقال من قواعد البيانات العلائقية العامة إلى النمذجة المتخصصة للكيانات البيولوجية. فهم الفرق بين الجين، والنسخة الرناوية (Transcript)، والبروتين، وكيفية تمثيل البيانات الوصفية للتسلسلات والتجارب في هيكل تطبيعي صارم. ثم إدراك متى وماذا نضحي بالتطبيع من أجل الأداء (Denormalization)، وتطبيق كل ذلك في مخطط متكامل لمشروع "Gene Resistance Tracker" الذي سيكون الأساس العملي لباقي الأبواب.

Purpose of this unit:

To move from general relational databases to specialized modeling of biological entities. To understand the difference between a gene, a transcript, and a protein, and how to represent sequence and experimental metadata in a rigorously normalized structure. Then to realize when and what to sacrifice in normalization for performance (Denormalization), and to apply all of this in an integrated Schema for the "Gene Resistance Tracker" project, which will serve as the practical foundation for the remaining units.

الفصل الثالث عشر: كيف نمثل الجين، النسخة الرناوية، والبروتين؟

Chapter Thirteen: How to Represent Gene, Transcript, and Protein?

الشكل 13 — الخريطة المفاهيمية للفصل

Figure 13 — Conceptual map of the chapter

الشكل 13 — سير العمل الهندسي للفصل

Figure 13 — Engineering workflow of the chapter

الأساس الذي تم بناؤه في الجزء الثاني يسمح لنا الآن بـ كان مفهوم SeqRecord يخلط أحياناً بين التسلسل الجينومي (DNA)، والنسخة الرناوية (mRNA/cDNA)، والتسلسل البروتيني. في قاعدة البيانات، يجب فصل هذه الكيانات لأن لها هويات وعلاقات مختلفة. الجين هو وحدة وراثية مجردة على الكروموسوم. النسخة الرناوية (Transcript) هي المنتج الفعلي لعملية النسخ من DNA إلى RNA، وقد ينتج جين واحد عدة نسخ عبر التضفير البديل (Alternative Splicing). البروتين هو المنتج النهائي لعملية الترجمة من RNA إلى سلسلة أحماض أمينية.

After working with objects and data in the previous part, the SeqRecord concept sometimes mixed genomic sequence (DNA), transcript (mRNA/cDNA),

and protein sequence. In a database, these entities must be separated because they have different identities and relationships. Gene is an abstract genetic unit on the chromosome. Transcript is the actual product of transcription from DNA to RNA; one gene may produce several transcripts via Alternative Splicing. Protein is the final product of translation from RNA to an amino acid chain.

العلاقات الأساسية (Cardinality):

Basic Relationships (Cardinality):

- الجين (Gene) → النسخ الرناوية (Transcripts) هي علاقة واحد إلى متعدد (1:N). جين واحد (مثل BRCA1) ينتج عدة نسخ (Isoforms).
- النسخة الرناوية (Transcript) → البروتين (Protein) هي علاقة واحد إلى واحد (1:1) في معظم الحالات (كل نسخة تترجم إلى بروتين واحد)، لكن بعض النسخ قد لا تترجم (Non-coding RNAs)، لذا يمكن أن تكون NULL.

· البروتين (Protein) → المسارات الحيوية (Pathways) هي علاقة متعدد إلى متعدد (M:N)، كما تعلمنا سابقاً.

تمثيل هذه الكيانات في SQL:

Representing these entities in SQL:

SQL

```
-- ============================================================
-- جدول الجينات (الكيان المجرد على الكروموسوم)
-- Genes table (abstract entity on chromosome)
-- ============================================================
CREATE TABLE genes (
    gene_id INTEGER PRIMARY KEY AUTOINCREMENT,
    gene_symbol TEXT UNIQUE NOT NULL,      -- مثل BRCA1, TP53
    chromosome TEXT NOT NULL,
    start_position INTEGER,                -- الموضع على الكروموسوم (1-based)
    end_position INTEGER,
    strand INTEGER NOT NULL CHECK(strand IN (1, -1)),
    description TEXT,
    source_db TEXT,                        -- مثل NCBI, Ensembl
    external_id TEXT UNIQUE,               -- معرف NCBI (مثل 672)
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);

-- فهارس لتسريع البحث حسب الرمز والكروموسوم
CREATE INDEX idx_genes_symbol ON genes(gene_symbol);
```

```sql
CREATE INDEX idx_genes_chromosome ON genes(chromosome);

-- ============================================================
-- جدول النسخ الرناوية (Transcripts) - المنتج الفعلي للنسخ
-- Transcripts table - actual transcription product
-- ============================================================
CREATE TABLE transcripts (
    transcript_id INTEGER PRIMARY KEY AUTOINCREMENT,
    gene_id INTEGER NOT NULL,
    transcript_symbol TEXT NOT NULL,      -- مثل BRCA1-201, BRCA1-202
    sequence_rna TEXT NOT NULL,           -- تسلسل الـ RNA (يحتوي على U بدلاً من T)
    length INTEGER,
    gc_content REAL CHECK(gc_content BETWEEN 0 AND 100),
    is_coding BOOLEAN DEFAULT 1,          -- هل يترجم إلى بروتين؟
    external_id TEXT UNIQUE,              -- معرف NCBI للـ transcript (مثل NM_007294)
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (gene_id) REFERENCES genes(gene_id) ON DELETE CASCADE
);

CREATE INDEX idx_transcripts_gene ON transcripts(gene_id);
CREATE INDEX idx_transcripts_symbol ON transcripts(transcript_symbol);

-- ============================================================
-- جدول البروتينات (Proteins) - المنتج النهائي للترجمة
-- Proteins table - final translation product
-- ============================================================
CREATE TABLE proteins (
    protein_id INTEGER PRIMARY KEY AUTOINCREMENT,
    transcript_id INTEGER UNIQUE NOT NULL, -- 1:1 relationship
    sequence_aa TEXT NOT NULL,            -- تسلسل الأحماض الأمينية (مثل MV...)
    length INTEGER,
    molecular_weight REAL,                -- الوزن الجزيئي (كيلو دالتون)
```

```sql
    isoelectric_point REAL,             -- نقطة التعادل الكهربي (pI)
    external_id TEXT UNIQUE,             -- معرف UniProt (مثل P38398)
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (transcript_id) REFERENCES transcripts(transcript_id) ON DELETE CASCADE
);

-- فهرس لتسريع البحث في تسلسل الأحماض الأمينية (في قواعد البيانات الكبيرة)
CREATE INDEX idx_proteins_transcript ON proteins(transcript_id);
```

ربط هذا بـ Biopython (الجزء الثاني):

Connecting this to Biopython (Part Two):

هنا يتحول ما تعلمته سابقًا إلى طبقة جديدة من التصميم، عند قراءة ملف GenBank باستخدام SeqIO.read()، كنا نحصل على SeqRecord واحد. هذا السجل قد يمثل الجين كاملاً (DNA) أو النسخة الرناوية (mRNA) أو البروتين (Protein) حسب نوع الملف. في نظامنا الجديد، سنقوم بتحويل (Mapping) كل نوع إلى الجدول المناسب:

- إذا كنا نتعامل مع ملف بروتين (FASTA بروتين)، نخزن في proteins.
- إذا كان record.annotations['molecule_type'] == 'DNA'، نخزن في genes أو transcripts حسب السياق.

· إذا كان record.annotations['molecule_type'] == 'RNA'، نخزن في transcripts.

هذا الفصل يسمح لنا بتتبع المصدر بدقة: أي بروتين جاء من أي نسخة، وأي نسخة جاءت من أي جين.

The foundation built in Part Two now allows you to when reading a GenBank file with SeqIO.read(), we obtained a single SeqRecord. This record might represent the entire gene (DNA), the transcript (mRNA), or the protein (Protein) depending on the file type. In our new system, we will map each type to the appropriate table:

· If record.annotations['molecule_type'] == 'DNA', we store in genes or transcripts depending on context.
· If record.annotations['molecule_type'] == 'RNA', we store in transcripts.
· If we are dealing with a protein file (Protein FASTA), we store in proteins.

This separation allows us to precisely track provenance: which protein came from which transcript, and which transcript came from which gene.

الفصل الرابع عشر: البيانات الوصفية للتسلسل — تجاوز النص الطويل

Chapter Fourteen: Sequence Metadata — Beyond the Long Text

الشكل 14 — الخريطة المفاهيمية للفصل

Figure 14 — Conceptual map of the chapter

الخبرة السابقة مع الملفات والكائنات تصبح في هذا الفصل مدخلًا إلى، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

الشكل 14 — سير العمل الهندسي للفصل

Figure 14 — Engineering workflow of the chapter

Here, an earlier skill becomes a new layer of system design, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

تخزين sequence_text كنص طويل هو الأساس، لكن التسلسل الحقيقي يحمل معه تاريخاً وسياقاً. البيانات الوصفية

(Metadata) هي ما يجعل التسلسل قابلاً للتفسير علمياً، وليس مجرد سلسلة من الحروف. في هذا الفصل، نضيف طبقات من البيانات الوصفية التي تجعل النظام متكاملاً وقابلاً للمراجعة.

Storing sequence_text as long text is foundational, but a real sequence carries history and context. Metadata is what makes a sequence scientifically interpretable, not just a string of letters. In this chapter, we add layers of metadata that make the system integrated and auditable.

14.1 البيانات الوصفية الأساسية للتسلسل

14.1 Basic Sequence Metadata.

- الطول (Length): محسوب من LENGTH(sequence_text). نخزنه لتجنب حساب الطول في كل استعلام (تحسين الأداء).

· نسبة GC (GC Content): محسوبة من النص، تمنحنا معلومات عن الاستقرار الحراري.

· نوع الجزيء (Molecule Type): RNA ،DNA، أو Protein.

· الكائن الحي المصدر (Source Organism): قد يكون Homo sapiens، E. coli، إلخ.

· رقم إصدار التجميع (Assembly Version): مثل GRCh38 للجينوم البشري.

Adding these fields to the sequences (or transcripts) table:

إضافة هذه الحقول إلى جدول sequences (أو transcripts):

SQL

```
-- توسيع جدول النسخ الرناوية (Transcripts) بمزيد من البيانات الوصفية
-- Extending the Transcripts table with more metadata
ALTER TABLE transcripts ADD COLUMN organism TEXT DEFAULT 'unknown';
ALTER TABLE transcripts ADD COLUMN assembly_version TEXT;
ALTER TABLE transcripts ADD COLUMN sequence_checksum TEXT; -- SHA-256 للتحقق من السلامة
ALTER TABLE transcripts ADD COLUMN source_db_version TEXT;
```

14.2 إصدار التسلسلات (Sequence Versioning) — تتبع التغييرات عبر الزمن

14.2 Sequence Versioning — Tracking Changes Over Time.

في الجزء الثاني (الفصل 53)، تحدثنا عن أن قواعد البيانات الحية مثل GenBank تتغير باستمرار. التسلسل الذي تحمله اليوم قد يُصحح غداً. كيف نتعامل مع هذا في قاعدة بياناتنا العلائقية؟ الإجابة هي التأريخ الثنائي (Bi-temporal Modeling) أو ببساطة إصدار التسلسل.

In Part Two (Chapter 53), we discussed that live databases like GenBank change constantly. The sequence you download today might be corrected tomorrow. How do we handle this in our relational database? The answer is Bi-temporal Modeling or simply sequence versioning.

نضيف عمودين إلى جدول transcripts:

· version: رقم الإصدار (1, 2, 3...).

· valid_from: التاريخ الذي أصبح فيه هذا الإصدار سارياً.

· valid_until: التاريخ الذي استُبدل فيه هذا الإصدار (أو NULL إذا كان هو الإصدار الحالي).

We add two columns to the transcripts table:

· version: Version number (1, 2, 3...).

· valid_from: The date this version became valid.

· valid_until: The date this version was superseded (or NULL if it is the current version).

SQL

```
-- إضافة إدارة الإصدارات إلى جدول النسخ الرناوية
-- Adding versioning to the Transcripts table
ALTER TABLE transcripts ADD COLUMN version INTEGER DEFAULT 1;
ALTER TABLE transcripts ADD COLUMN valid_from TEXT DEFAULT CURRENT_TIMESTAMP;
```

```
ALTER TABLE transcripts ADD COLUMN valid_until TEXT; -- NULL يعني أنه الإصدار الحالي

-- عند إدراج إصدار جديد، نغلق القديم --
-- When inserting a new version, we close the old one
-- مثال على تحديث (يتم برمجياً في الـ Repository):
-- UPDATE transcripts SET valid_until = CURRENT_TIMESTAMP WHERE transcript_symbol =
'BRCA1-201' AND valid_until IS NULL;
-- INSERT INTO transcripts (... , version=2, valid_from=CURRENT_TIMESTAMP) ... ;
```

ربط هذا بـ OOP (الجزء العملي):

Connecting this to OOP (Practical):

Python

```python
from dataclasses import dataclass
from typing import Optional

@dataclass
class Transcript:
    transcript_id: int
    gene_id: int
    transcript_symbol: str
    sequence_rna: str
    length: int
    gc_content: float
    is_coding: bool
    organism: str
    version: int
    valid_from: str
    valid_until: Optional[str] = None
```

```
def is_current(self) -> bool:
    return self.valid_until is None
```

الفصل الخامس عشر: البيانات الوصفية للتجارب — سياق توليد البيانات

Chapter Fifteen: Experimental Metadata — The Context of Data Generation

الشكل 15 — الخريطة المفاهيمية للفصل

Figure 15 — Conceptual map of the chapter

الشكل 15 — سير العمل الهندسي للفصل

Figure 15 — Engineering workflow of the chapter

التسلسل لا يُولد في الفراغ. يأتي من تجربة معملية محددة: عينة، بروتوكول، جهاز تسلسل، تاريخ، ومشغل. هذه البيانات الوصفية التجريبية هي ما يسمح لنا بتقييم جودة البيانات وفهم مصادر التباين (Batch Effects) التي تحدثنا عنها في الجزء الثاني (RNA-seq).

A sequence is not generated in a vacuum. It comes from a specific laboratory experiment: sample, protocol, sequencing machine, date, and operator. This experimental metadata is what allows us to assess data quality and understand sources of variation (Batch Effects) we discussed in Part Two (RNA-seq).

15.1 هيكل البيانات الوصفية للتجارب

15.1 Experimental Metadata Structure.

SQL

```
-- ============================================================
-- جدول التجارب (Experiments) - يربط العينات بالبروتوكولات
-- Experiments table - links samples to protocols
-- ============================================================
CREATE TABLE experiments (
    experiment_id INTEGER PRIMARY KEY AUTOINCREMENT,
    sample_id INTEGER NOT NULL,
    protocol_name TEXT NOT NULL,         -- مثلا "RNA-seq polyA"
    protocol_version TEXT,
    library_type TEXT CHECK(library_type IN ('stranded', 'unstranded')),
    target_read_length INTEGER,          -- طول القراءة المستهدف (مثل 150)
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sample_id) REFERENCES samples(sample_id) ON DELETE CASCADE
```

```
);

-- ============================================================
-- جدول التشغيلات (Runs) - التشغيل الفعلي على جهاز التسلسل
-- Runs table - actual run on the sequencer
-- ============================================================
CREATE TABLE runs (
    run_id INTEGER PRIMARY KEY AUTOINCREMENT,
    experiment_id INTEGER NOT NULL,
    instrument_model TEXT NOT NULL,        -- مثل NovaSeq 6000, MinION
    run_date TEXT,
    total_reads INTEGER,                   -- عدد القراءات الإجمالي
    total_bases INTEGER,                   -- عدد القواعد الإجمالي
    q30_percent REAL CHECK(q30_percent BETWEEN 0 AND 100), -- نسبة الجودة > Q30
    fastq_path TEXT,                       -- مسار ملف FASTQ (أو رابط)
    checksum TEXT,                         -- SHA-256 لملف FASTQ
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (experiment_id) REFERENCES experiments(experiment_id) ON DELETE
CASCADE
);

-- فهارس
CREATE INDEX idx_runs_experiment ON runs(experiment_id);
CREATE INDEX idx_runs_date ON runs(run_date);
```

15.2 ربط البيانات الوصفية للتجارب بالتسلسلات (التحليل النهائي)

15.2 Linking Experimental Metadata to Sequences (Final Analysis).

الآن، يمكننا ربط كل تسلسل (في جدول transcripts أو sequences) بتشغيل محدد، وبالتالي ببروتوكول وعينة ومريض. هذا يسمح لنا بتتبع سلسلة القياس كاملة: من المريض إلى الحمض النووي إلى القراءات الخام إلى التسلسل المُجمّع.

Now, we can link each sequence (in transcripts or sequences) to a specific run, and thus to a protocol, sample, and patient. This allows us to trace the entire measurement chain: from the patient to the DNA to the raw reads to the assembled sequence.

SQL

```sql
-- ربط التسلسل مباشرة بالتشغيل (بدلاً من العينة فقط)
-- Linking sequence directly to the run (instead of just the sample)
ALTER TABLE sequences ADD COLUMN run_id INTEGER;
ALTER TABLE sequences ADD COLUMN alignment_score REAL; -- درجة المحاذاة
ALTER TABLE sequences ADD COLUMN mapped_reads INTEGER; -- عدد القراءات المحاذية

ALTER TABLE sequences ADD FOREIGN KEY (run_id) REFERENCES runs(run_id);
```

الفصل السادس عشر: التطبيع (**Normalization**) — لماذا تقسيم الجداول؟

Chapter Sixteen: Normalization — Why Split Tables?

الشكل 16 — الخريطة المفاهيمية للفصل

Figure 16 — Conceptual map of the chapter

الآن نستخدم المهارة السابقة للوصول إلى مستوى جديد من النظام. بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

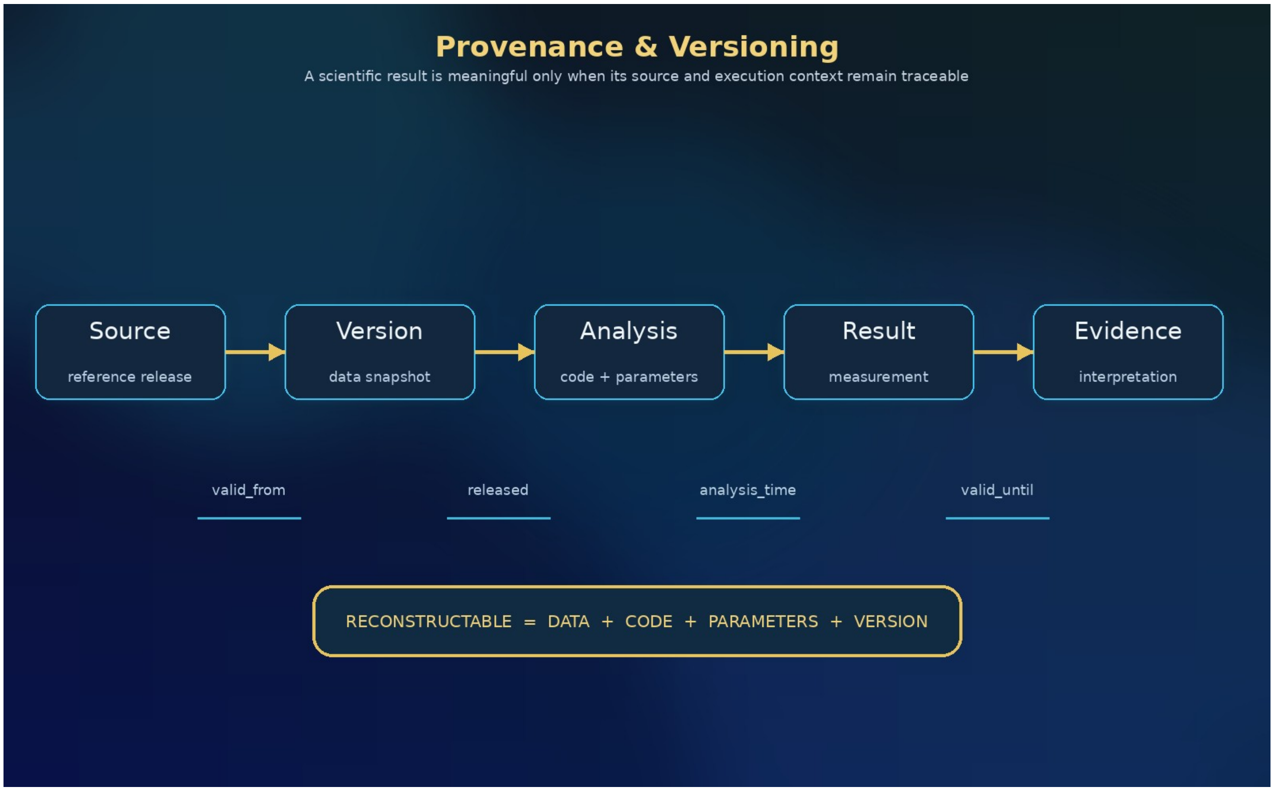
الشكل 16 — سير العمل الهندسي للفصل

Figure 16 — Engineering workflow of the chapter

16.5 البيانات ثنائية الزمن وإدارة الإصدارات

Your previous experience with files and objects now becomes an entry point to you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

16.5 Bi-Temporal Data and Version Management.



سلسلة المصدر والإصدار والتحليل والنتيجة التي تجعل النتيجة قابلة لإعادة البناء.

The source, version, analysis, and result chain that makes a result reconstructable.

البيانات البيولوجية ليست ثابتة. قد يتغير التعليق على تسلسل، أو تتغير حالة سجل، أو يظهر إصدار أحدث من قاعدة مرجعية. لذلك لا يكفي أن تحفظ القيمة الحالية؛ في المشاريع

القابلة لإعادة الإنتاج تحتاج إلى معرفة أي إصدار كان صالحًا، ومتى استُخدم في التحليل.

Biological data are not static. An annotation may change, a record may be revised, or a newer release of a reference database may appear. Therefore, storing only the current value is insufficient; reproducible projects need to know which version was valid and which version was used for an analysis.

استخدم حقولًا زمنية واضحة مثل valid_from وvalid_until، وأضف إلى سجل التحليل معرف الإصدار الذي دخل فعليًا في الحساب. لا تخلط بين وقت صلاحية المعلومة في العالم العلمي وبين وقت إدخالها إلى قاعدة البيانات؛ الفصل بينهما هو الفكرة الأساسية في البيانات ثنائية الزمن.

Use explicit temporal fields such as valid_from and valid_until, and attach the exact source version used by an analysis to the analysis record. Do not confuse the time at which a fact was valid in the scientific world with the time at which it was entered into the database; separating these dimensions is the core idea of bi-temporal data.

SQL

```
CREATE TABLE sequence_versions (
    version_id INTEGER PRIMARY KEY,
    accession TEXT NOT NULL,
    sequence TEXT NOT NULL,
    valid_from TEXT NOT NULL,
    valid_until TEXT,
    source_release TEXT NOT NULL
);

CREATE TABLE analysis_runs (
    analysis_id INTEGER PRIMARY KEY,
    version_id INTEGER NOT NULL,
    executed_at TEXT NOT NULL,
    FOREIGN KEY (version_id)
        REFERENCES sequence_versions(version_id)
);
```

في الفصول السابقة، قمنا بتقسيم البيانات إلى جداول متعددة (genes، transcripts، proteins، samples، patients). هذا التقسيم يسمى التطبيع (Normalization). الهدف هو التخلص من التكرار والحفاظ على سلامة البيانات. التطبيع له أشكال (Normal Forms) متعددة، لكننا سنركز على الثلاثة الأولى التي تهم المصمم العملي.

In previous chapters, we divided data into multiple tables (genes, transcripts, proteins, samples, patients). This division is called Normalization. The goal is to eliminate redundancy and maintain data integrity. Normalization has multiple Normal Forms, but we will focus on the first three that matter to the practical designer.

16.1 الشكل الطبيعي الأول (1NF) — الذرية

16.1 First Normal Form (1NF) — Atomicity.

القاعدة: كل عمود يجب أن يحتوي على قيم ذرية (غير قابلة للتجزئة)، ويجب ألا تحتوي الخلية على مجموعة من القيم.

Rule: Every column must contain atomic (indivisible) values, and a cell must not contain a set of values.

مخالفة 1NF: تخزين عدة جينات في عمود واحد مفصولة بفواصل.

Violation of 1NF: Storing multiple genes in one column separated by commas.

SQL

```
-- خطأ: عمود gene_symbols يحتوي على قائمة
CREATE TABLE bad_samples (sample_id INT, gene_symbols TEXT); -- 'BRCA1,TP53,EGFR'
```

الحل (1NF): جدول sequences منفصل، كل صف يحتوي على جين واحد.

Solution (1NF): Separate sequences table, each row contains one gene.

16.2 الشكل الطبيعي الثاني (2NF) — الاعتماد الكامل على المفتاح الأساسي

16.2 Second Normal Form (2NF) — Full Dependency on the Primary Key.

القاعدة: يجب أن يعتمد كل عمود غير مفتاحي على المفتاح الأساسي بالكامل، وليس على جزء منه (ينطبق على المفاتيح المركبة).

Rule: Every non-key column must depend on the entire Primary Key, not just part of it (applies to composite keys).

مثال: لنفترض أن لدينا جدول sample_measurements بمفتاح مركب (sample_id, gene_id) يحتوي أيضاً على patient_age. patient_age يعتمد فقط على sample_id (الذي يرتبط بـ patient_id)، وليس على gene_id. هذا ينتهك 2NF.

Example: Suppose we have a sample_measurements table with a composite key (sample_id, gene_id) also containing patient_age. patient_age depends only on sample_id (which links to patient_id), not on gene_id. This violates 2NF.

الحل: نقل patient_age إلى جدول patients، وربطه عبر samples (كما فعلنا).

Solution: Move patient_age to the patients table, linked via samples (as we did).

16.3 الشكل الطبيعي الثالث (3NF) — لا اعتماد انتقالي

16.3 Third Normal Form (3NF) — No Transitive Dependency.

القاعدة: يجب ألا يعتمد عمود غير مفتاحي على عمود غير مفتاحي آخر (اعتماد انتقالي).

Rule: A non-key column must not depend on another non-key column (transitive dependency).

مثال: جدول transcripts يحتوي على gene_symbol و gene_description. gene_description يعتمد على gene_symbol، و gene_symbol يعتمد على gene_id (المفتاح). هذا اعتماد انتقالي.

Example: transcripts table contains gene_symbol and gene_description. gene_description depends on gene_symbol, and gene_symbol depends on gene_id (the key). This is a transitive dependency.

الحل: نقل gene_symbol و gene_description إلى جدول genes، وترك gene_id فقط في transcripts. هذا ما فعلناه، حيث أن transcripts تحتوي فقط على gene_id كمفتاح أجنبي.

Solution: Move gene_symbol and gene_description to the genes table, leaving only gene_id in transcripts. This is what we did, as transcripts only contains gene_id as a foreign key.

الفصل السابع عشر: إزالة التطبيع (Denormalization) — متى نضحي بالقواعد من أجل الأداء؟

Chapter Seventeen: Denormalization — When Do We Sacrifice Rules for Performance?

الشكل 17 — الخريطة المفاهيمية للفصل

Figure 17 — Conceptual map of the chapter

في هذه المرحلة من المسار، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

الشكل 17 — سير العمل الهندسي للفصل

Figure 17 — Engineering workflow of the chapter

We now use the earlier skill to reach the next level of the system. you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

التطبيع الصارم ينتج عنه العديد من الجداول والكثير من عمليات JOIN في الاستعلامات. في النظام الحيوي الكبير، قد يكون الأداء بطيئاً. إزالة التطبيع (Denormalization) هي عملية إضافة تكرار محسوب إلى قاعدة البيانات لتسريع القراءة، مع قبول تحديات الصيانة الإضافية. هذا هو المقايضة الهندسية (Engineering Trade-off) الكلاسيكية: مساحة تخزين مقابل سرعة.

Strict normalization results in many tables and many JOIN operations in queries. In a large biological system, performance might be slow. Denormalization is the process of adding calculated redundancy to the database to speed up reading, accepting the maintenance challenges. This is the classic Engineering Trade-off: storage space versus speed.

17.1 أمثلة على Denormalization في المعلوماتية الحيوية

17.1 Examples of Denormalization in Bioinformatics.

1. تخزين متوسط التعبير (Calculated Aggregates):

بدلاً من حساب AVG(expression) كل مرة عبر JOIN مع جدول ضخم للقراءات، نخزن avg_expression في جدول genes ونحدثه دورياً (أو عبر Triggers).

1. Storing Calculated Aggregates:

Instead of calculating AVG(expression) every time via JOIN with a huge reads table, we store avg_expression in the genes table and update it periodically (or via Triggers).

SQL

```
-- إضافة عمود مشتق إلى جدول الجينات لتسريع لوحات القيادة (Dashboards)
ALTER TABLE genes ADD COLUMN avg_expression REAL;
ALTER TABLE genes ADD COLUMN max_expression REAL;
```

2. دمج جداول متكررة الاستعلام (Flattening Frequent Joins):

إذا كنا نستعلم دائماً عن transcript_symbol مع gene_symbol، يمكننا إضافة gene_symbol (وهو تكرار) إلى جدول transcripts.

2. Flattening Frequent Joins:

If we always query transcript_symbol with gene_symbol, we can add gene_symbol (which is a duplication) to the transcripts table.

SQL

```
-- إضافة عمود مكرر لتجنب JOIN مع جدول genes في الاستعلامات البسيطة
ALTER TABLE transcripts ADD COLUMN gene_symbol TEXT;
-- لكن يجب الحفاظ على التزامن مع جدول genes (عبر Trigger أو كود التطبيق)
```

17.2 متى نستخدم Denormalization؟

17.2 When to Use Denormalization?

- عندما تكون القراءة أكثر تكراراً من الكتابة: (نظام قراءة تقارير، لوحات تحكم).

· عندما تكون جداول JOIN ضخمة: (استعلام يتطلب ربط 5 جداول كبيرة).

· عندما تكون البيانات "شبه ثابتة": (البيانات المرجعية مثل الكروموسومات والأنواع تتغير نادراً).

· When reads are far more frequent than writes: (Reporting systems, Dashboards).

· When JOIN tables are massive: (A query requires joining 5 large tables).

· عندما تكون البنية التحتية محدودة: (ذاكرة قليلة، وقت استجابة حرج).

· When data is "semi-static": (Reference data like chromosomes and species rarely changes).

· When infrastructure is limited: (Low memory, critical response time).

تحذير هندسي: كل Denormalization يجب أن يكون موثقاً بشكل صريح. يجب أن تعرف لماذا كررت البيانات. قاعدة بيانات

مصممة بشكل سيء مع Denormalization عشوائي تصبح كابوساً في الصيانة.

Engineering Warning: Every Denormalization must be explicitly documented. You must know why you duplicated data. A poorly designed database with random Denormalization becomes a maintenance nightmare.

الفصل الثامن عشر: تصميم قاعدة بيانات متكاملة لمشروع "Gene Resistance Tracker"

Chapter Eighteen: Designing an Integrated Database for the "Gene Resistance Tracker" Project

الشكل 18 — الخريطة المفاهيمية للفصل

Figure 18 — Conceptual map of the chapter

الشكل 18 — سير العمل الهندسي للفصل

Figure 18 — Engineering workflow of the chapter

عند الانتقال من الجزء السابق إلى هذا الفصل، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

At this stage of the path, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

18.5 The Gene Resistance Tracker — The System-Wide Thread.

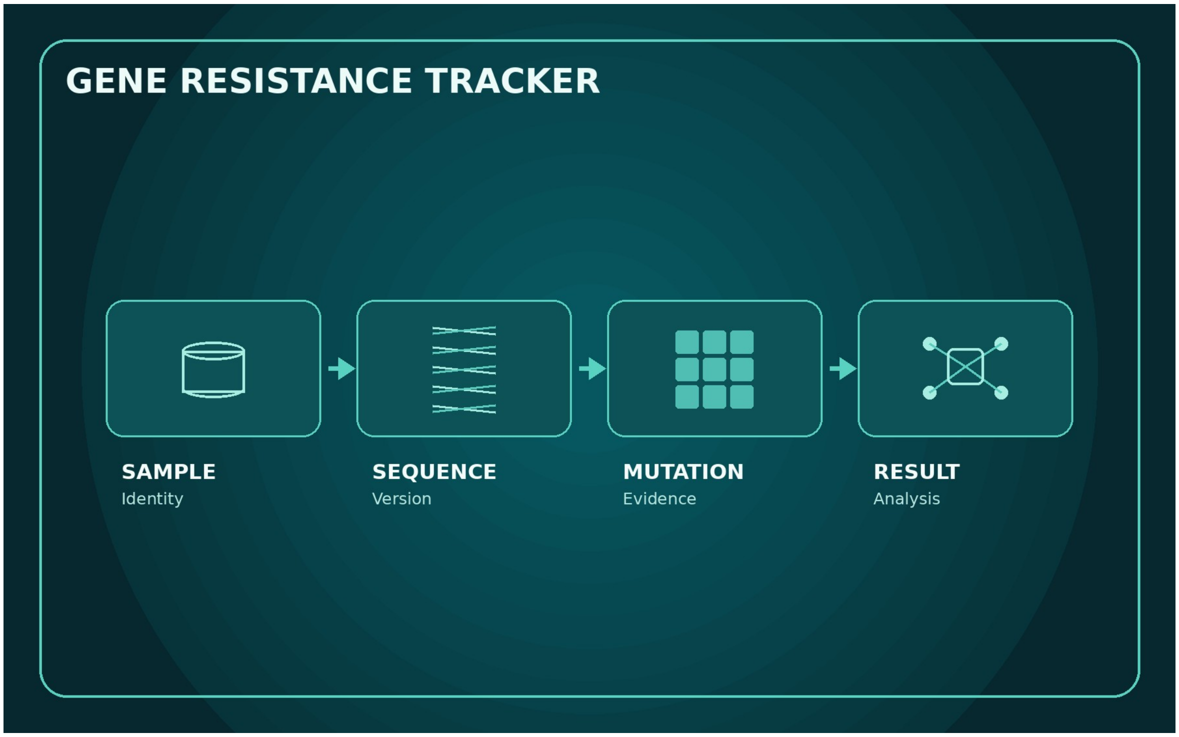


Figure — GENE RESISTANCE TRACKER

18.5 مشروع Gene Resistance Tracker — الخيط الموحد للنظام

من هذه النقطة فصاعدًا، تعامل مع Gene Resistance Tracker كمشروع واحد يتطور معك. يبدأ المشروع بسؤال علمي: كيف نربط الجينات والطفرات والبيانات التجريبية بملاحظات مقاومة موثقة دون فقدان المصدر أو الإصدار؟ ثم يتحول السؤال إلى نموذج كيانات، ثم Schema، ثم SQL، ثم طبقة OOP،

ثم تحليل عددي، ثم جزء كمومي عند وجود مسألة مناسبة فعلًا.

From this point onward, treat the Gene Resistance Tracker as one evolving project. It begins with a scientific question: how can genes, mutations, experimental data, and documented resistance observations be connected without losing provenance or version information? The question becomes an entity model, then a Schema, then SQL, then an OOP layer, then numerical analysis, and finally a quantum component only when a suitable problem genuinely exists.

SQL

```
CREATE TABLE resistance_observations (
    observation_id INTEGER PRIMARY KEY,
    gene_id INTEGER NOT NULL,
    mutation_id INTEGER,
    phenotype TEXT NOT NULL,
    evidence_level TEXT NOT NULL,
    source_version TEXT NOT NULL,
    FOREIGN KEY (gene_id) REFERENCES genes(gene_id),
    FOREIGN KEY (mutation_id) REFERENCES mutations(mutation_id)
);
```

كل باب سيضيف طبقة إلى هذا المشروع بدل أن يبدأ مثالًا منفصلًا. بهذه الطريقة يصبح انتقالك بين SQL وOOP والجبر الخطي والحوسبة الكمومية انتقالًا داخل نظام واحد.

Each unit will add a layer to this project instead of starting an unrelated example. This makes the transition between SQL, OOP, linear algebra, and quantum computing a transition inside one system.

هذا هو الفصل التطبيقي الذي يجمع كل ما تعلمناه في الباب الرابع. سنقوم بتصميم مخطط (Schema) متكامل للمشروع العملي، مع تطبيق التطبيع المناسب، وإضافة بعض عناصر Denormalization المحسوبة حيث نرى أنها مفيدة للوحات التحكم (Dashboards) والاستعلامات السريعة. هذا المخطط سيكون الأساس الذي نبني عليه طبقة Python (الباب الخامس) وسير العمل (الباب السادس).

This is the applied chapter that brings together everything we learned in Unit Four. We will design an integrated Schema for the practical project, applying appropriate Normalization, and adding some calculated Denormalization elements where we see they benefit Dashboards and fast queries. This Schema will be the foundation for the Python layer (Unit Five) and the workflow (Unit Six).

18.1 المخطط المتكامل الكامل

18.1 The Complete Integrated Schema.

SQL

```
-- ============================================================
-- مشروع: Gene Resistance Tracker - المخطط المتكامل
-- Project: Gene Resistance Tracker - Integrated Schema
-- ============================================================


-- ----------------------------
-- 1. الطبقة السريرية (Clinical Layer)
-- ----------------------------
CREATE TABLE patients (
    patient_id INTEGER PRIMARY KEY AUTOINCREMENT,
    external_id TEXT UNIQUE NOT NULL,
    age INTEGER CHECK(age >= 0 AND age < 150),
```

```
    sex TEXT CHECK(sex IN ('M', 'F', 'O')),
    diagnosis TEXT,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE samples (
    sample_id INTEGER PRIMARY KEY AUTOINCREMENT,
    patient_id INTEGER NOT NULL,
    tissue_type TEXT NOT NULL,
    collection_date TEXT,
    rna_quality REAL CHECK(rna_quality BETWEEN 0 AND 10),
    status TEXT CHECK(status IN ('control', 'diseased', 'treated')),
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id) ON DELETE CASCADE
);
CREATE INDEX idx_samples_patient ON samples(patient_id);

-- ---------------------------
-- 2. الطبقة التجريبية (Experimental Layer)
-- ---------------------------
CREATE TABLE experiments (
    experiment_id INTEGER PRIMARY KEY AUTOINCREMENT,
    sample_id INTEGER NOT NULL,
    protocol_name TEXT NOT NULL,
    library_type TEXT CHECK(library_type IN ('stranded', 'unstranded')),
    target_read_length INTEGER,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sample_id) REFERENCES samples(sample_id) ON DELETE CASCADE
);
CREATE INDEX idx_experiments_sample ON experiments(sample_id);

CREATE TABLE runs (
```

```
    run_id INTEGER PRIMARY KEY AUTOINCREMENT,
    experiment_id INTEGER NOT NULL,
    instrument_model TEXT NOT NULL,
    run_date TEXT,
    total_reads INTEGER,
    total_bases INTEGER,
    q30_percent REAL CHECK(q30_percent BETWEEN 0 AND 100),
    fastq_path TEXT,
    checksum TEXT,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (experiment_id) REFERENCES experiments(experiment_id) ON DELETE
CASCADE
);
CREATE INDEX idx_runs_experiment ON runs(experiment_id);

-- ----------------------------
-- 3. الطبقة الجزيئية (Molecular Layer) - التطبيع الكامل
-- ----------------------------
CREATE TABLE genes (
    gene_id INTEGER PRIMARY KEY AUTOINCREMENT,
    gene_symbol TEXT UNIQUE NOT NULL,
    chromosome TEXT NOT NULL,
    start_position INTEGER,
    end_position INTEGER,
    strand INTEGER NOT NULL CHECK(strand IN (1, -1)),
    description TEXT,
    external_id TEXT UNIQUE,
    -- Denormalization: حقول مشتقة لتسريع لوحات التحكم
    avg_expression REAL,
    max_expression REAL,
    mutation_count INTEGER DEFAULT 0,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
```

```
);
CREATE INDEX idx_genes_symbol ON genes(gene_symbol);

CREATE TABLE transcripts (
    transcript_id INTEGER PRIMARY KEY AUTOINCREMENT,
    gene_id INTEGER NOT NULL,
    transcript_symbol TEXT NOT NULL,
    sequence_rna TEXT NOT NULL,
    length INTEGER,
    gc_content REAL CHECK(gc_content BETWEEN 0 AND 100),
    is_coding BOOLEAN DEFAULT 1,
    organism TEXT DEFAULT 'unknown',
    assembly_version TEXT,
    version INTEGER DEFAULT 1,
    valid_from TEXT DEFAULT CURRENT_TIMESTAMP,
    valid_until TEXT, -- NULL = current
    external_id TEXT UNIQUE,
    FOREIGN KEY (gene_id) REFERENCES genes(gene_id) ON DELETE CASCADE
);
CREATE INDEX idx_transcripts_gene ON transcripts(gene_id);

CREATE TABLE proteins (
    protein_id INTEGER PRIMARY KEY AUTOINCREMENT,
    transcript_id INTEGER UNIQUE NOT NULL,
    sequence_aa TEXT NOT NULL,
    length INTEGER,
    molecular_weight REAL,
    external_id TEXT UNIQUE,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (transcript_id) REFERENCES transcripts(transcript_id) ON DELETE CASCADE
);
CREATE INDEX idx_proteins_transcript ON proteins(transcript_id);
```

```sql
-- ---------------------------
-- 4. الطبقة الوظيفية (Functional Layer) - الطفرات والمقاومة
-- ---------------------------
CREATE TABLE mutations (
    mutation_id INTEGER PRIMARY KEY AUTOINCREMENT,
    transcript_id INTEGER NOT NULL,      -- نربط الطفرة بالنسخة، وليس بالتسلسل الجينومي
    position INTEGER NOT NULL,
    ref_base TEXT NOT NULL CHECK(ref_base IN ('A', 'T', 'C', 'G', 'U')),
    alt_base TEXT NOT NULL CHECK(alt_base IN ('A', 'T', 'C', 'G', 'U')),
    effect TEXT CHECK(effect IN ('missense', 'nonsense', 'silent', 'frameshift')),
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (transcript_id) REFERENCES transcripts(transcript_id) ON DELETE CASCADE,
    UNIQUE (transcript_id, position)
);
CREATE INDEX idx_mutations_transcript ON mutations(transcript_id);

CREATE TABLE resistance (
    resistance_id INTEGER PRIMARY KEY AUTOINCREMENT,
    drug_name TEXT UNIQUE NOT NULL,
    drug_class TEXT,
    description TEXT
);

-- الجدول الوسيط M:N
CREATE TABLE mutation_resistance (
    mutation_id INTEGER NOT NULL,
    resistance_id INTEGER NOT NULL,
    confidence REAL DEFAULT 1.0 CHECK(confidence BETWEEN 0 AND 1),
    evidence TEXT,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (mutation_id, resistance_id),
```

```
    FOREIGN KEY (mutation_id) REFERENCES mutations(mutation_id) ON DELETE CASCADE,
    FOREIGN KEY (resistance_id) REFERENCES resistance(resistance_id) ON DELETE CASCADE
);
CREATE INDEX idx_mr_mutation ON mutation_resistance(mutation_id);
CREATE INDEX idx_mr_resistance ON mutation_resistance(resistance_id);


-- ----------------------------
-- 5. الطبقة الإدارية (Administrative Layer) - الأمن والمراجعة
-- ----------------------------
CREATE TABLE users (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE NOT NULL,
    role TEXT CHECK(role IN ('admin', 'researcher', 'auditor')) DEFAULT 'researcher',
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE audit_log (
    audit_id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    action TEXT NOT NULL,
    table_name TEXT NOT NULL,
    record_id TEXT,
    old_value TEXT,
    new_value TEXT,
    timestamp TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE SET NULL
);
CREATE INDEX idx_audit_table ON audit_log(table_name);
CREATE INDEX idx_audit_time ON audit_log(timestamp);
```

18.2 ربط المخطط بـ OOP (تمثيل الكائنات المتطابقة)

18.2 Connecting the Schema to OOP (Representing Matching Objects).

سنقوم بإنشاء كائنات Python @dataclass تمثل الصفوف في هذه الجداول، مع الحفاظ على التسلسل الهرمي (Gene يحتوي على Transcripts، Transcript يحتوي على Protein و Mutations). هذا يعكس بنية OOP التي تعلمناها في الجزء الثاني، لكن مع دعم الاستمرارية (Persistence).

We will create Python @dataclass objects representing rows in these tables, maintaining the hierarchy (Gene contains Transcripts, Transcript contains Protein and Mutations). This reflects the OOP structure we learned in Part Two, but with Persistence support.

Python

```
from dataclasses import dataclass, field
from typing import List, Optional
```

```
@dataclass
class Protein:
    protein_id: int
    sequence_aa: str
    length: int
    molecular_weight: Optional[float] = None
    external_id: Optional[str] = None

@dataclass
class Mutation:
    mutation_id: int
    position: int
    ref_base: str
    alt_base: str
    effect: Optional[str] = None

@dataclass
class Transcript:
    transcript_id: int
    transcript_symbol: str
    sequence_rna: str
    length: int
    gc_content: float
    is_coding: bool
    organism: str
    version: int
    valid_from: str
    valid_until: Optional[str] = None
    protein: Optional[Protein] = None  # 1:1 relation
    mutations: List[Mutation] = field(default_factory=list)  # 1:N relation

@dataclass
```

```
class Gene:
    gene_id: int
    gene_symbol: str
    chromosome: str
    strand: int
    description: Optional[str] = None
    avg_expression: Optional[float] = None  # Denormalized
    mutation_count: int = 0  # Denormalized
    transcripts: List[Transcript] = field(default_factory=list)  # 1:N relation
```

ملاحظة: هذه الكائنات تعكس البنية المنطقية. في طبقة المستودع (Repository)، سنقوم بتحويل هذه الكائنات إلى صفوف والعكس، مع JOIN لجلب transcripts عند استدعاء find_by_id لجين، تماماً كما كنا نفعل مع features في SeqRecord في الجزء الثاني، لكن الآن على مستوى قاعدة البيانات.

Note: These objects reflect the logical structure. In the Repository layer, we will convert these objects to rows and vice versa, using JOIN to fetch transcripts when calling find_by_id for a gene, just as we did with features in SeqRecord in Part Two, but now at the database level.

الانتقال إلى الباب الخامس:

Transition to Unit Five:

المخطط جاهز، والكائنات OOP جاهزة. حان الوقت لبناء الجسر الحقيقي بينهما: طبقة البيانات (Data Layer) باستخدام نمط الـ Repository، مع الاتصال الفعلي بقاعدة البيانات، وكتابة دوال save و find الواقعية التي تتعامل مع العلاقات المعقدة (حفظ جين كامل مع نسخه وطفراته في معاملة واحدة (Transaction) أو عدة معاملات مترابطة). الباب الخامس سيجعل SQL و OOP يعملان معاً كفريق واحد.

The Schema is ready, and the OOP objects are ready. It is time to build the real bridge between them: the Data Layer using the Repository pattern, with actual database connection, and writing realistic save and find functions that handle complex relationships (saving a complete Gene with its transcripts and

mutations in a single Transaction or several interrelated ones). Unit Five will make SQL and OOP work together as one team.

—

الباب الخامس: **SQL + Python + OOP** —— طبقة البيانات الهندسية

Unit Five: SQL + Python + OOP — The Engineering Data Layer

الغرض من هذا الباب:

بناء الجسر البرمجي الفعلي بين كائنات OOP التي صممناها في الباب الرابع وجداول SQL التي أنشأناها في نفس الباب. هذا هو الباب الذي يجعل الجزء الثالث امتداداً طبيعياً للجزء الثاني، حيث تتحول الكائنات من مجرد هياكل في الذاكرة إلى كيانات دائمة في قاعدة البيانات. سنطبق نمط المستودع (Repository Pattern) بشكل متكامل، مع التعامل مع المعاملات (Transactions)، وسلامة البيانات (Integrity)، والتحقق من

الصحة (Validation)، وبناء طبقة بيانات قابلة لإعادة الاستخدام والاختبار.

Purpose of this unit:

To build the actual programmatic bridge between the OOP objects designed in Unit Four and the SQL tables created in the same unit. This is the unit that makes Part Three a natural extension of Part Two, where objects transition from mere in-memory structures to persistent entities in a database. We will implement the Repository Pattern comprehensively, handling Transactions, Data Integrity, Validation, and building a reusable, testable data layer.

الفصل التاسع عشر: الاتصال بقاعدة البيانات من Python — sqlite3 كأساس عملي

Chapter Nineteen: Connecting to the Database from Python — sqlite3 as a Practical Foundation

الاتصال بقاعدة البيانات جزء من هندسة الموارد، وليس مجرد سطر .import التطبيق الجيد يحدد بوضوح من يفتح الاتصال، ومن يغلقه، وما عمر الاتصال، ومتى تبدأ المعاملة ومتى تُثبت أو تُلغى. وفي SQLite يجب أن تفكر أيضًا في التزامن، وقفل الملف، وحدود الكتابة المتزامنة، وحجم قاعدة البيانات. هذه التفاصيل هي التي تفصل المثال التعليمي الذي يعمل مرة واحدة عن طبقة بيانات يمكن الاعتماد عليها داخل تطبيق علمي طويل التشغيل.

Database connectivity is a resource-management concern, not merely an import statement. A robust application defines who opens the connection, who closes it, how long the connection lives, and when a transaction begins, commits, or rolls back. With SQLite you must also reason about concurrency, file locking, concurrent writes, and the practical limits of database size. These details distinguish a one-off educational example from a data layer that can be trusted inside a long-running scientific application.

مختبر: حد اتصال آمن بين Python وقاعدة البيانات

Laboratory: A Safe Python Database Boundary

يجب أن تتحكم طبقة Python في عمر الاتصال وتمثيل الصفوف. وتفيد الصفوف المسماة عندما تحتوي النتيجة على حقول كثيرة لأنها تجعل الربط بين التخزين وكائنات المجال واضحًا.

The Python layer should control connection lifetime and row representation. Named rows are valuable when a result contains many fields because they make the mapping from storage to domain objects explicit.

لا تدمج القيم غير الموثوقة داخل نصوص SQL. مررها كمعلمات من خلال sqlite3 أو أي تنفيذ آخر لـDB-API.

Never concatenate untrusted values into SQL strings. Bind them as parameters through sqlite3 or another DB-API implementation.

وصول قابل لإعادة الاستخدام ومعلَّم بالمعلمات

Reusable and parameterized database access

```
import sqlite3
```

```
class Database:
    def __init__(self, path: str):
        self.path = path

    def samples_for_organism(self, organism: str):
        with sqlite3.connect(self.path) as con:
            con.row_factory = sqlite3.Row
            return con.execute(
                "SELECT sample_id, collection_date FROM samples WHERE organism = ? ORDER BY
collection_date",
                (organism,),
            ).fetchall()
```

PYTHON

الشكل 19 —– الخريطة المفاهيمية للفصل

Figure 19 — Conceptual map of the chapter

المعرفة التي بُنيت في الجزء الثاني تقودنا هنا إلى أن، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

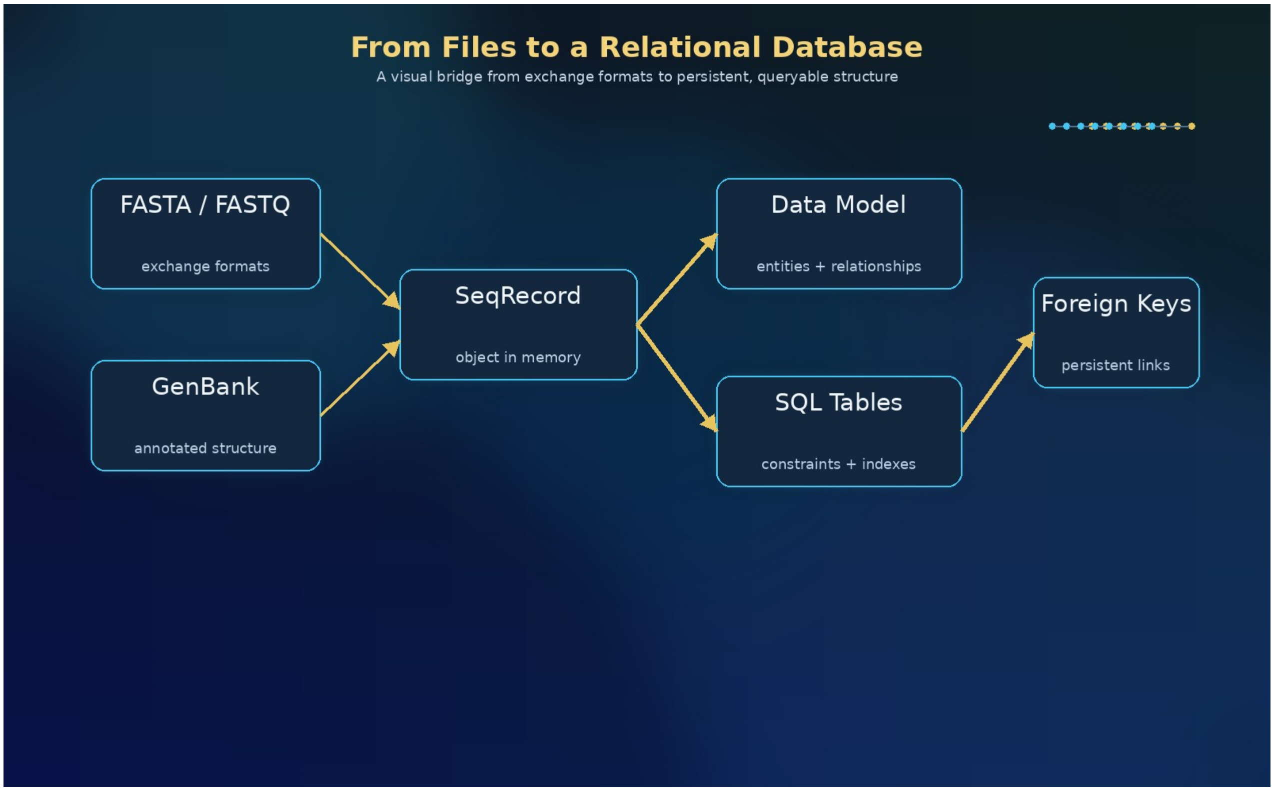
الشكل 19 —– سير العمل الهندسي للفصل

Figure 19 — Engineering workflow of the chapter

19.5 من الكائن المتداخل إلى الجدول المسطح —– جسر Biopython إلى SQL

As you move from the previous part into this chapter, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

19.5 From a Nested Object to Flat Tables — The Biopython-to-SQL Bridge.



انتقال الكائن الحيوي الغني من الذاكرة إلى نموذج علائقي دائم.

A visual bridge from a rich biological object in memory to a persistent relational model.

يمثل SeqRecord في Biopython تسلسلًا مع هوية ووصف وتعليقات وميزات. هذه البنية ممتازة داخل Python، لكنها لا تعني أن قاعدة البيانات يجب أن تخزن الكائن كاملًا ككتلة واحدة. الأفضل أن تفصل المعلومات التي تحتاج إلى الاستعلام المستقل عنها إلى جداول مترابطة.

A Biopython SeqRecord represents a sequence together with identity, description, annotations, and features. This structure is excellent inside Python, but it does not mean that a database should store the entire object as one opaque block. Information that must be queried independently should be separated into related tables.

توثيق Biopython يوضح أن SeqRecord هو طبقة أعلى من Seq وأن SeqFeature تحمل معلومات الميزات المرتبطة بالتسلسل.

The Biopython documentation explains that SeqRecord is a higher-level sequence representation and that SeqFeature objects carry annotated features associated with a sequence.

Python

```
from dataclasses import dataclass
```

```python
from typing import List


@dataclass
class Feature:
    start: int
    end: int
    feature_type: str


@dataclass
class SequenceRecord:
    accession: str
    sequence: str
    features: List[Feature]
```

SQL

```sql
CREATE TABLE sequence_records (
    sequence_id INTEGER PRIMARY KEY,
    accession TEXT NOT NULL UNIQUE,
    sequence TEXT NOT NULL
);


CREATE TABLE sequence_features (
    feature_id INTEGER PRIMARY KEY,
    sequence_id INTEGER NOT NULL,
    start_pos INTEGER NOT NULL,
    end_pos INTEGER NOT NULL,
    feature_type TEXT NOT NULL,
    FOREIGN KEY (sequence_id) REFERENCES sequence_records(sequence_id)
);
```

قبل أن نكتب أي استعلام معقد، يجب أن نفهم كيفية اتصال Python بقاعدة البيانات. في هذا الكتاب، نستخدم sqlite3 لأنه مدمج في Python ولا يحتاج إلى تثبيت خادم منفصل، مما يجعله مثالياً للتعليم والتطوير الأولي. لكن المبادئ التي سنتعلمها (Connection، Cursor، Transaction، Parameterized Queries) تنطبق على أي قاعدة بيانات علائقية (PostgreSQL، MySQL، إلخ).

Before writing any complex query, we must understand how Python connects to a database. In this book, we use sqlite3 because it is built into Python and requires no separate server installation, making it ideal for education and initial development. However, the principles we will learn (Connection, Cursor, Transaction, Parameterized Queries) apply to any relational database (PostgreSQL, MySQL, etc.).

19.1 الاتصال الأساسي (Connection) والمؤشر (Cursor)

19.1 Basic Connection and Cursor.

Python

```
import sqlite3
from pathlib import Path

# تحديد مسار قاعدة البيانات
db_path = Path("genomic_data.db")

# الاتصال بقاعدة البيانات (تنشئ الملف إذا لم يكن موجوداً)
connection = sqlite3.connect(db_path)

# إنشاء مؤشر (Cursor) لتنفيذ الأوامر
cursor = connection.cursor()

# تنفيذ استعلام بسيط
cursor.execute("SELECT sqlite_version();")
version = cursor.fetchone()
print(f"SQLite version: {version[0]}")

# إغلاق الاتصال (مهم لتحرير الموارد)
connection.close()
```

الممارسة الهندسية الجيدة: استخدام with لضمان إغلاق الاتصال تلقائياً.

Good Engineering Practice: Use with to ensure the connection is automatically closed.

Python

```
with sqlite3.connect("genomic_data.db") as connection:
   cursor = connection.cursor()
   cursor.execute("SELECT sqlite_version();")
   print(cursor.fetchone()[0])
 #  يتم إغلاق الاتصال تلقائياً عند الخروج من with
 #  Connection is automatically closed when exiting the with block
```

19.2 الاستعلامات المحددة (Parameterized Queries) — منع حقن SQL

19.2 Parameterized Queries — Preventing SQL Injection.

خطأ شائع: دمج المتغيرات مباشرة في نص الاستعلام.

Common Mistake: Concatenating variables directly into the query string.

Python

```
 #  خطير: عرضة لحقن SQL
```

```
# Dangerous: Vulnerable to SQL Injection
gene = "BRCA1"
cursor.execute(f"SELECT * FROM genes WHERE gene_symbol = '{gene}'")
```

الحل الصحيح: استخدام ? كعناصر نائبة (Placeholders) وتمرير القيم كـ Tuple.

Correct Solution: Use ? as Placeholders and pass values as a Tuple.

Python

```
# آمن: استخدام العناصر النائبة
# Safe: Using placeholders
gene = "BRCA1"
cursor.execute("SELECT * FROM genes WHERE gene_symbol = ?", (gene,))
rows = cursor.fetchall()
```

19.3 إدارة المعاملات (Transactions) — العمليات الذرية

19.3 Transaction Management — Atomic Operations.

في sqlite3، المعاملات تبدأ تلقائياً عند أول أمر INSERT أو UPDATE أو DELETE، وتُحفظ باستخدام commit() أو تُلغي باستخدام rollback().

In sqlite3, transactions start automatically upon the first INSERT, UPDATE, or DELETE command, and are saved using commit() or cancelled using rollback().

Python

```
def save_gene_with_transcript(gene_data: dict, transcript_data: dict) -> bool:
    """
    حفظ جين ونسخته في معاملة واحدة (All or Nothing).
    Save a gene and its transcript in a single transaction (All or Nothing).
    """
    connection = None
    try:
        connection = sqlite3.connect("genomic_data.db")
        cursor = connection.cursor()

        # إدراج الجين
        cursor.execute(
            "INSERT INTO genes (gene_symbol, chromosome, strand) VALUES (?, ?, ?)",
            (gene_data["symbol"], gene_data["chromosome"], gene_data["strand"])
        )
        gene_id = cursor.lastrowid
```

```
        # إدراج النسخة الرناوية (Transcript) المرتبطة بالجين
        cursor.execute(
            "INSERT INTO transcripts (gene_id, transcript_symbol, sequence_rna, length, is_coding) VALUES (?, ?, ?, ?, ?)",
            (gene_id, transcript_data["symbol"], transcript_data["rna"], transcript_data["length"], transcript_data["is_coding"])
        )

        # حفظ جميع التغييرات
        connection.commit()
        return True

    except sqlite3.IntegrityError as e:
        print(f"خطأ في سلامة البيانات: {e}")
        if connection:
            connection.rollback()  # التراجع عن جميع التغييرات
        return False

    except Exception as e:
        print(f"خطأ غير متوقع: {e}")
        if connection:
            connection.rollback()
        return False

    finally:
        if connection:
            connection.close()
```

الفصل العشرون: تحويل الصفوف إلى كائنات — إعادة بناء OOP من SQL

Chapter Twenty: Converting Rows to Objects — Rebuilding OOP from SQL

مختبر: إعادة بناء كائنات المجال من الصفوف

Laboratory: Reconstructing Domain Objects from Rows

يمثل الصف التخزين، بينما يمثل كائن المجال المعنى. وعند إعادة بناء الكائن يمكن جعل التحقق والثوابت والسلوك صريحة مرة أخرى.

A row represents storage. A domain object represents meaning. Rebuilding the object is where validation, invariants, and behavior can become explicit again.

استخدم كائنات قيمة غير قابلة للتغيير للقياسات الصغيرة عندما يؤدي التغيير العرضي إلى تغيير المعنى العلمي.

Use immutable value objects for small measurements when accidental mutation would change scientific meaning.

إعادة بناء كائن المجال

Rebuilding a domain object

```
from dataclasses import dataclass


@dataclass(frozen=True)
class MutationResult:
    mutation_id: int
    gene_symbol: str
    evidence_level: str


def to_result(row):
    return MutationResult(
        mutation_id=row['mutation_id'],
        gene_symbol=row['gene_symbol'],
        evidence_level=row['evidence_level'],
    )
```

PYTHON

الشكل 20 — الخريطة المفاهيمية للفصل

Figure 20 — Conceptual map of the chapter

عندما يصبح المشروع أكبر، يمكن استخدام ORM كطبقة تجريد إضافية. SQLAlchemy يعرّف نماذج Python مرتبطة بجداول ويتيح

إنشاء الجداول والاستعلام عنها، مع بقاء فهم SQL نفسه ضروريًا لأن ORM لا يلغي النموذج العلاقي.

الشكل 20 — سير العمل الهندسي للفصل

Figure 20 — Engineering workflow of the chapter

As a project grows, an ORM can provide an additional abstraction layer. SQLAlchemy maps Python models to tables and supports table creation and querying, while understanding SQL remains essential because an ORM does not eliminate relational design.

Python

```
from sqlalchemy import String, create_engine
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column, Session


class Base(DeclarativeBase):
    pass


class GeneModel(Base):
    __tablename__ = "genes"

    gene_id: Mapped[int] = mapped_column(primary_key=True)
    symbol: Mapped[str] = mapped_column(String(50), unique=True)
    organism: Mapped[str] = mapped_column(String(100))


engine = create_engine("sqlite:///genes.db")
Base.metadata.create_all(engine)
```

```
with Session(engine) as session:
    session.add(GeneModel(symbol="BRCA1", organism="Homo sapiens"))
    session.commit()
```

من الأدوات التي اكتسبتها في الجزء الثاني، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

The knowledge built in Part Two leads directly here because you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

الآن بعد أن أصبح لدينا اتصال بقاعدة البيانات، نحتاج إلى ترجمة الصفوف (Rows) التي تعيدها SQL إلى كائنات OOP التي نتعامل معها في التطبيق. هذه العملية تسمى تعيين النتائج إلى كائنات (Result-to-Object Mapping).

Now that we have a database connection, we need to translate the Rows returned by SQL into OOP objects we work with in the application. This process is called Result-to-Object Mapping.

20.1 استخدام row_factory لتحويل الصفوف إلى قاموس

20.1 Using row_factory to Convert Rows to Dictionaries.

sqlite3.Row يسمح لنا بالوصول إلى الأعمدة بالاسم (مثل القاموس) بدلاً من الفهرس الرقمي. هذا يجعل الكود أكثر قراءة وأقل عرضة للأخطاء عند تغيير ترتيب الأعمدة.

sqlite3.Row allows us to access columns by name (like a dictionary) instead of by numeric index. This makes the code more readable and less error-prone when column order changes.

Python

```
def get_gene_by_symbol(gene_symbol: str):
    with sqlite3.connect("genomic_data.db") as connection:
        connection.row_factory = sqlite3.Row
        cursor = connection.cursor()
        cursor.execute(
            "SELECT gene_id, gene_symbol, chromosome, strand, description FROM genes WHERE
gene_symbol = ?",
            (gene_symbol,)
        )
        row = cursor.fetchone()
        if row:
            # الوصول بالأعمدة بالاسم
```

```
        return {
            "id": row["gene_id"],
            "symbol": row["gene_symbol"],
            "chromosome": row["chromosome"],
            "strand": row["strand"],
            "description": row["description"]
        }
    return None
```

20.2 تحويل الصفوف إلى كائنات @dataclass

20.2 Converting Rows to @dataclass Objects.

نحن نستخدم @dataclass لتعريف الكائنات، مما يسهل إنشاءها من الصفوف.

We use @dataclass to define objects, making it easy to create them from rows.

Python

```
from dataclasses import dataclass
from typing import Optional


@dataclass
```

```
class Gene:
    gene_id: int
    gene_symbol: str
    chromosome: str
    strand: int
    description: Optional[str] = None
    avg_expression: Optional[float] = None

@dataclass
class Transcript:
    transcript_id: int
    gene_id: int
    transcript_symbol: str
    sequence_rna: str
    length: int
    is_coding: bool
    gc_content: Optional[float] = None

def row_to_gene(row: sqlite3.Row) -> Gene:
    """تحويل صف SQLite إلى كائن Gene."""
    return Gene(
        gene_id=row["gene_id"],
        gene_symbol=row["gene_symbol"],
        chromosome=row["chromosome"],
        strand=row["strand"],
        description=row.get("description"),
        avg_expression=row.get("avg_expression")
    )

def row_to_transcript(row: sqlite3.Row) -> Transcript:
    """تحويل صف SQLite إلى كائن Transcript."""
    return Transcript(
```

```
        transcript_id=row["transcript_id"],
        gene_id=row["gene_id"],
        transcript_symbol=row["transcript_symbol"],
        sequence_rna=row["sequence_rna"],
        length=row["length"],
        is_coding=bool(row["is_coding"]),
        gc_content=row.get("gc_content")
    )

# استخدام عملية التحويل في استعلام
def get_gene_with_transcripts(gene_symbol: str) -> Optional[Gene]:
    with sqlite3.connect("genomic_data.db") as connection:
        connection.row_factory = sqlite3.Row
        cursor = connection.cursor()

        # جلب بيانات الجين
        cursor.execute(
            "SELECT * FROM genes WHERE gene_symbol = ?",
            (gene_symbol,)
        )
        gene_row = cursor.fetchone()
        if not gene_row:
            return None

        gene = row_to_gene(gene_row)

        # جلب النسخ المرتبطة
        cursor.execute(
            "SELECT * FROM transcripts WHERE gene_id = ?",
            (gene.gene_id,)
        )
        transcript_rows = cursor.fetchall()
```

```
    gene.transcripts = [row_to_transcript(r) for r in transcript_rows]


    return gene
```

20.3 التعامل مع العلاقات 1:N — تحميل الكائنات المرتبطة

20.3 Handling 1:N Relationships — Loading Related Objects.

في المثال أعلاه، قمنا بتحميل transcripts المرتبطة بـ gene في استعلام منفصل. هذا يعرف بـ التحميل الكسول (Lazy Loading) إذا حدث عند الطلب، أو التحميل المسبق (Eager Loading) إذا حدث فوراً.

In the example above, we loaded the transcripts associated with the gene in a separate query. This is known as Lazy Loading if it happens on demand, or Eager Loading if it happens immediately.

للعلاقات 1:N المتعمقة (Gene → Transcripts → Mutations)، يمكننا تحميل كل شيء في استعلام واحد باستخدام JOIN، لكننا سنفقد البنية الهرمية. البديل الأفضل هو استعلامات متعددة منظمة (كما فعلنا) أو استخدام CTEs مع JSON (في SQLite 3.38 +).

For deep 1:N relationships (Gene → Transcripts → Mutations), we can load everything in one query using JOIN, but we lose the hierarchical structure. The better alternative is multiple structured queries (as we did) or using CTEs with JSON (in SQLite 3.38+).

Python

```
def get_full_gene(gene_symbol: str) -> Optional[Gene]:
    """جلب الجين مع جميع النسخ والطفرات المرتبطة (تحميل مسبق كامل).
    Fetch gene with all associated transcripts and mutations (Full Eager Loading).
    """
    with sqlite3.connect("genomic_data.db") as connection:
        connection.row_factory = sqlite3.Row
        cursor = connection.cursor()

        # 1. الجين
        cursor.execute("SELECT * FROM genes WHERE gene_symbol = ?", (gene_symbol,))
        gene_row = cursor.fetchone()
```

```
if not gene_row:
    return None
gene = row_to_gene(gene_row)

# 2. النسخ
cursor.execute("SELECT * FROM transcripts WHERE gene_id = ?", (gene.gene_id,))
transcript_rows = cursor.fetchall()
transcripts = [row_to_transcript(r) for r in transcript_rows]

# 3. الطفرات لكل نسخة
for transcript in transcripts:
    cursor.execute(
        "SELECT * FROM mutations WHERE transcript_id = ?",
        (transcript.transcript_id,)
    )
    mutation_rows = cursor.fetchall()
    transcript.mutations = [row_to_mutation(r) for r in mutation_rows]

gene.transcripts = transcripts
return gene
```

الفصل الحادي والعشرون: نمط المستودع (**Repository**) — فصل منطق التخزين

Chapter Twenty-One: The Repository Pattern — Separating Storage Logic

مختبر: تصميم مستودع موجه نحو المجال

Laboratory: Domain-Oriented Repository Design

يجب أن يعرض المستودع عمليات تعكس الهدف العلمي. يبقى SQL داخل المستودع، بينما تتحدث طبقة الخدمة بلغة المجال.

A repository should expose operations that reflect scientific intent. SQL remains inside the repository, while the service layer speaks in domain terms.

قسّم المستودع عندما يكبر بحسب التجميع أو المسؤولية بدل إنشاء مستودع عالمي واحد.

Split a growing repository by aggregate or responsibility instead of creating one universal repository.

مستودع موجه نحو المجال

Domain-oriented repository

```
class MutationRepository:
    def __init__(self, con):
        self.con = con

    def resistance_for_gene(self, gene_symbol: str):
        sql = """
        SELECT mutation_id, hgvs, evidence_level
        FROM resistance_mutations
        WHERE gene_symbol = ?
        ORDER BY evidence_level DESC, hgvs
        """
        return self.con.execute(sql, (gene_symbol,)).fetchall()
```

PYTHON

الشكل 21 —— الخريطة المفاهيمية للفصل

Figure 21 — Conceptual map of the chapter

بعد أن أصبحت الأساسيات السابقة مستقرة، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

الشكل 21 —— سير العمل الهندسي للفصل

Figure 21 — Engineering workflow of the chapter

Among the tools you gained in Part Two, you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

نمط المستودع (Repository) هو واجهة برمجية بين منطق التطبيق (الذي يتعامل مع OOP) ومنطق التخزين (الذي يتعامل مع SQL). المستودع يخفي تفاصيل قاعدة البيانات عن باقي التطبيق. هذا يسمح لنا بتغيير قاعدة البيانات لاحقاً (مثلاً من SQLite إلى PostgreSQL) دون تعديل كود التطبيق.

The Repository Pattern is a software interface between application logic (which deals with OOP) and storage logic (which deals with SQL). The repository hides database details from the rest of the application. This allows us to change the database later (e.g., from SQLite to PostgreSQL) without modifying application code.

21.1 واجهة المستودع الأساسية (Abstract Base Class)

21.1 Basic Repository Interface (Abstract Base Class).

Python

```
from abc import ABC, abstractmethod
from typing import List, Optional

class GeneRepositoryInterface(ABC):
    """واجهة مجردة لمستودع الجينات.
    Abstract interface for the Gene repository.
    """

    @abstractmethod
    def save(self, gene: Gene) -> Gene:
        """حفظ جين (إدراج أو تحديث). إرجاع الكائن مع المعرف المحدث.
        Save a gene (insert or update). Returns the object with updated ID.
        """
        pass

    @abstractmethod
    def find_by_id(self, gene_id: int) -> Optional[Gene]:
        """البحث عن جين بواسطة المعرف.
        Find a gene by ID.
        """
        pass

    @abstractmethod
    def find_by_symbol(self, symbol: str) -> Optional[Gene]:
        """البحث عن جين بواسطة الرمز (مع النسخ).
        Find a gene by symbol (with transcripts).
```

```
        """
        pass

    @abstractmethod
    def find_all(self, limit: int = 100, offset: int = 0) -> List[Gene]:
        """استرجاع قائمة من الجينات (دون النسخ للتبسيط).
        Retrieve a list of genes (without transcripts for simplicity).
        """
        pass

    @abstractmethod
    def delete(self, gene_id: int) -> bool:
        """حذف جين بالكامل (مع النسخ والطفرات بفضل ON DELETE CASCADE).
        Delete a gene completely (with transcripts and mutations thanks to ON DELETE CASCADE).
        """
        pass
```

21.2 التطبيق العملي للمستودع باستخدام SQLite

21.2 Practical Implementation of the Repository using SQLite.

Python

```
class SqliteGeneRepository(GeneRepositoryInterface):
    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path
        self._ensure_schema()
```

```
def _ensure_schema(self) -> None:
    """التأكد من وجود الجداول (تم إنشاؤها في الباب الرابع).
    Ensure tables exist (created in Unit Four).
    """
    # تم حذف كود إنشاء الجداول للاختصار، لكنه موجود في الباب الرابع
    # Table creation code omitted for brevity, but exists in Unit Four
    pass

def save(self, gene: Gene) -> Gene:
    """حفظ جين مع نسخه وطفراته في معاملة واحدة.
    Save a gene with its transcripts and mutations in one transaction.
    """
    with sqlite3.connect(self.db_path) as connection:
        cursor = connection.cursor()

        # 1. حفظ الجين
        if gene.gene_id == 0:  # جديد
            cursor.execute(
                """INSERT INTO genes
                  (gene_symbol, chromosome, strand, description)
                  VALUES (?, ?, ?, ?)""",
                (gene.gene_symbol, gene.chromosome, gene.strand, gene.description)
            )
            gene.gene_id = cursor.lastrowid
        else:  # تحديث
            cursor.execute(
                """UPDATE genes
                  SET gene_symbol=?, chromosome=?, strand=?, description=?
                  WHERE gene_id=?""",
                (gene.gene_symbol, gene.chromosome, gene.strand,
                 gene.description, gene.gene_id)
            )
```

```
 #  2. حذف النسخ القديمة (للتحديث الكامل)
cursor.execute("DELETE FROM transcripts WHERE gene_id = ?", (gene.gene_id,))

 #  3. حفظ النسخ الجديدة والطفرات المرتبطة
for transcript in gene.transcripts:
    cursor.execute(
        """INSERT INTO transcripts
           (gene_id, transcript_symbol, sequence_rna, length,
            is_coding, gc_content, organism)
           VALUES (?, ?, ?, ?, ?, ?, ?)""",
        (gene.gene_id, transcript.transcript_symbol,
         transcript.sequence_rna, transcript.length,
         1 if transcript.is_coding else 0,
         transcript.gc_content, transcript.organism)
    )
    transcript_id = cursor.lastrowid

     #  حفظ طفرات النسخة
    for mutation in transcript.mutations:
        cursor.execute(
            """INSERT INTO mutations
               (transcript_id, position, ref_base, alt_base, effect)
               VALUES (?, ?, ?, ?, ?)""",
            (transcript_id, mutation.position,
             mutation.ref_base, mutation.alt_base,
             mutation.effect)
        )

connection.commit()
return gene
```

```
def find_by_symbol(self, symbol: str) -> Optional[Gene]:
    """استرجاع جين مع جميع نسخه وطفراته.
    Retrieve a gene with all its transcripts and mutations.
    """
    with sqlite3.connect(self.db_path) as connection:
        connection.row_factory = sqlite3.Row
        cursor = connection.cursor()

        # 1. جلب الجين
        cursor.execute(
            "SELECT * FROM genes WHERE gene_symbol = ?",
            (symbol,)
        )
        gene_row = cursor.fetchone()
        if not gene_row:
            return None

        gene = Gene(
            gene_id=gene_row["gene_id"],
            gene_symbol=gene_row["gene_symbol"],
            chromosome=gene_row["chromosome"],
            strand=gene_row["strand"],
            description=gene_row["description"],
            avg_expression=gene_row.get("avg_expression")
        )

        # 2. جلب النسخ
        cursor.execute(
            "SELECT * FROM transcripts WHERE gene_id = ?",
            (gene.gene_id,)
        )
        transcript_rows = cursor.fetchall()
```

```
for tr in transcript_rows:
    transcript = Transcript(
        transcript_id=tr["transcript_id"],
        gene_id=tr["gene_id"],
        transcript_symbol=tr["transcript_symbol"],
        sequence_rna=tr["sequence_rna"],
        length=tr["length"],
        is_coding=bool(tr["is_coding"]),
        gc_content=tr.get("gc_content"),
        organism=tr.get("organism", "unknown")
    )

    # 3. جلب الطفرات المرتبطة
    cursor.execute(
        "SELECT * FROM mutations WHERE transcript_id = ?",
        (transcript.transcript_id,)
    )
    mutation_rows = cursor.fetchall()
    transcript.mutations = [
        Mutation(
            mutation_id=m["mutation_id"],
            position=m["position"],
            ref_base=m["ref_base"],
            alt_base=m["alt_base"],
            effect=m.get("effect")
        ) for m in mutation_rows
    ]

    gene.transcripts.append(transcript)

return gene
```

```
def delete(self, gene_id: int) -> bool:
    """حذف جين. ON DELETE CASCADE يتعامل مع النسخ والطفرات.
    Delete a gene. ON DELETE CASCADE handles transcripts and mutations.
    """
    with sqlite3.connect(self.db_path) as connection:
        cursor = connection.cursor()
        cursor.execute("DELETE FROM genes WHERE gene_id = ?", (gene_id,))
        connection.commit()
        return cursor.rowcount > 0
```

الفصل الثاني والعشرون: المعاملات (**Transactions**) وسلامة البيانات

Chapter Twenty-Two: Transactions and Data Integrity

مختبر: كتابات علمية ذرية

Laboratory: Atomic Scientific Writes

يجب أن تلتزم الكتابة العلمية متعددة الخطوات كحدث منطقي واحد. فإذا فشل إدراج تابع، يجب أن تعود قاعدة البيانات إلى الحالة المتسقة السابقة.

A multi-step scientific write should commit as one logical event. If one dependent insert fails, the database should return to the previous consistent state.

يمكن بدء معاملات SQLite صراحةً ثم الالتزام بها أو التراجع عنها كوحدة واحدة. وهذه جزء من سلامة البيانات وليست مجرد راحة برمجية.

SQLite transactions can be started explicitly and committed or rolled back as a unit. This is part of data integrity, not merely convenience.

كتابة ذرية للبيانات والمصدر

Atomic write of data plus provenance

```python
with sqlite3.connect('genomic.db') as con:
    con.execute('BEGIN')
    con.execute(
        'INSERT INTO sequence_versions(accession, version, sequence) VALUES (?, ?, ?)',
        (accession, version, sequence),
    )
    con.execute(
        'INSERT INTO provenance(accession, source_release) VALUES (?, ?)',
        (accession, source_release),
    )
```

PYTHON

الشكل 22 — الخريطة المفاهيمية للفصل

Figure 22 — Conceptual map of the chapter

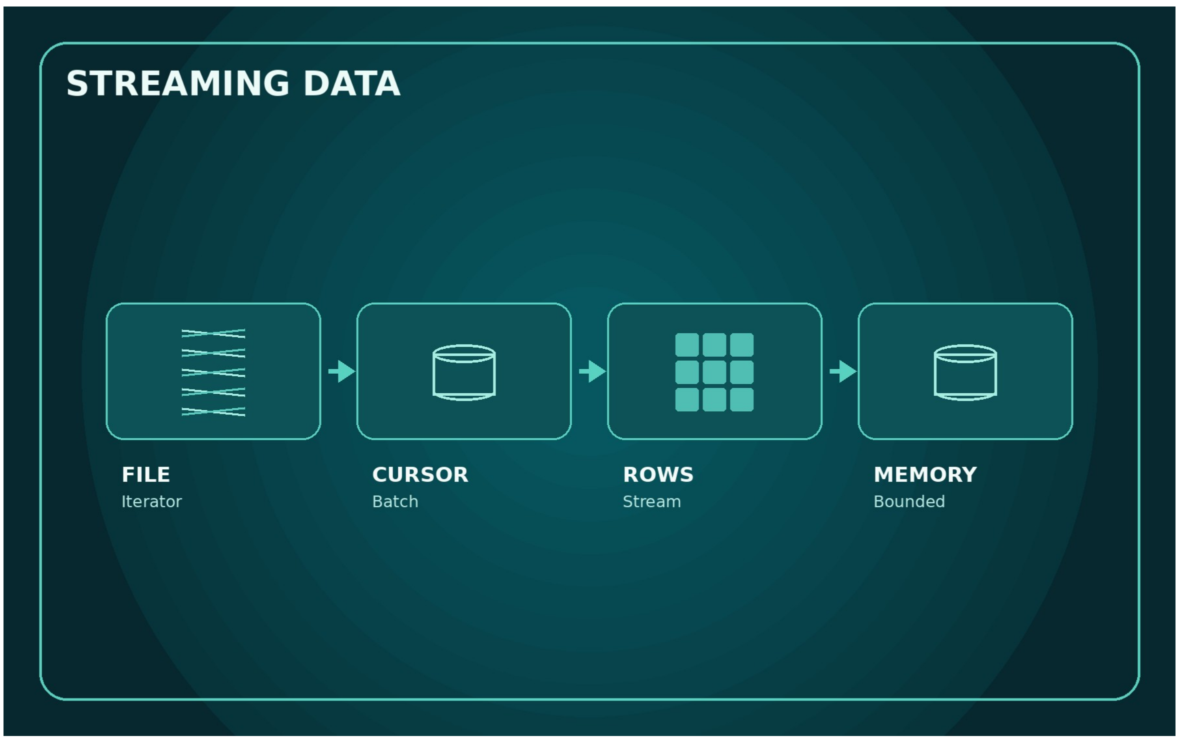
الشكل 22 — سير العمل الهندسي للفصل

Figure 22 — Engineering workflow of the chapter

ما تعلّمته سابقًا عن البيانات والبرمجة يصبح هنا نقطة انطلاق، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

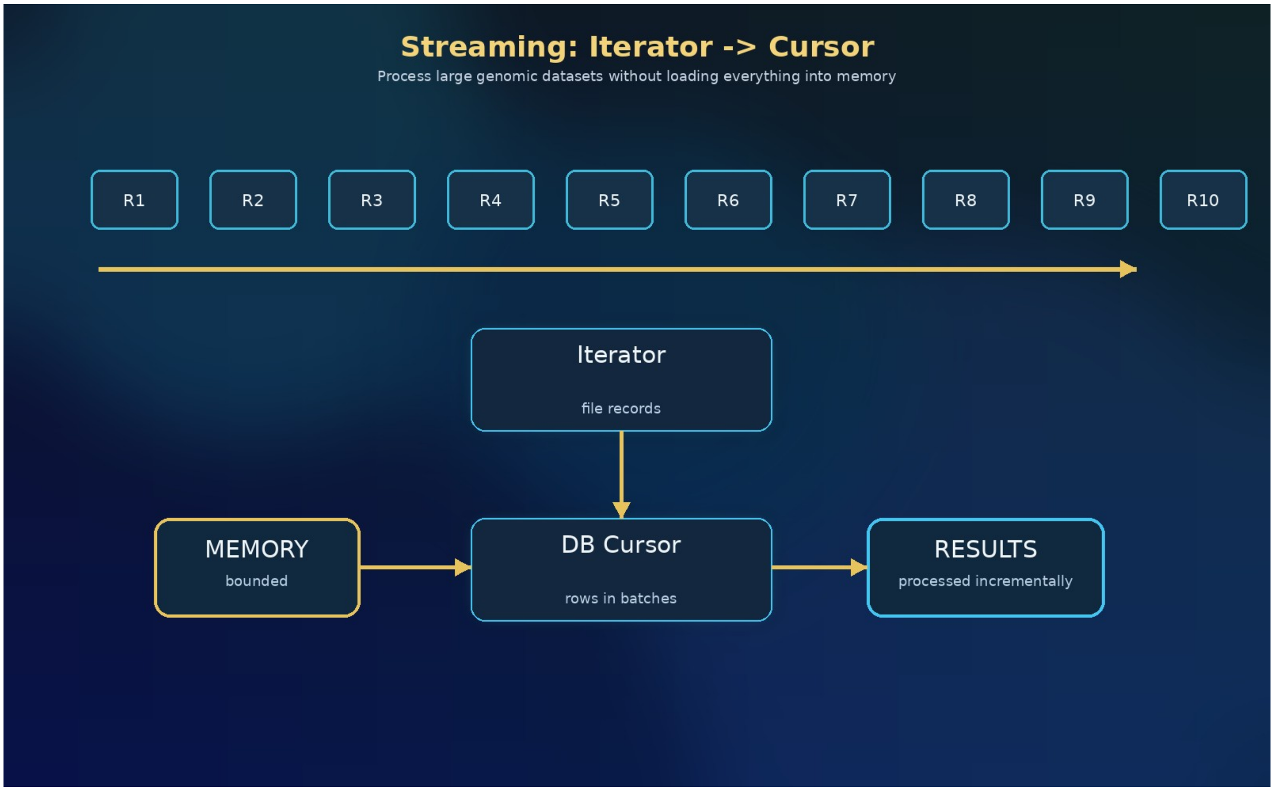
Once the earlier foundations are stable, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

22.5 From Python Iterators to Database Cursors — Streaming Without Consuming Memory.



22.5 من مكررات Python إلى مؤشرات قاعدة البيانات — Streaming دون ابتلاع الذاكرة

Figure — STREAMING DATA



المعالجة التدفقية تحافظ على الذاكرة بينما تمر البيانات سجلًا بعد سجل.

عندما تقرأ ملايين السجلات، لا يكون الهدف أن تجمع كل شيء في قائمة واحدة. الفكرة هي المعالجة المتتابعة: احصل على جزء صغير، عالجه، ثم انتقل إلى الجزء التالي. SeqIO.parse يعلّمك هذا النمط على الملفات، بينما cursor يتيح نمطًا مشابهًا عند قراءة نتائج قاعدة البيانات.

Streaming keeps memory bounded while records are processed one at a time. When you process millions of records, the goal is not to place everything into one list. The pattern is sequential processing: obtain a small portion, process it, and move to the next portion. SeqIO.parse teaches this pattern for files, while a database cursor provides a similar approach for query results.

Python

```
import sqlite3


def stream_sequences(db_path: str):
    with sqlite3.connect(db_path) as conn:
        cursor = conn.execute(
            "SELECT sequence_id, accession, sequence FROM sequences"
        )
        for row in cursor:
            yield row
```

لا تجعل cursor مرادفًا كاملًا لـiterator؛ إنهما مفهومان مختلفان، لكنهما يشتركان في نمط الوصول التدريجي إلى البيانات. هذا الفرق الدقيق مهم عندما تنتقل من الذاكرة المحلية إلى أنظمة بيانات أكبر.

Do not treat a cursor as a perfect synonym for an iterator; they are distinct concepts, but they share the pattern of incremental access to data. This distinction matters when you move from local memory to larger data systems.

المعاملات (Transactions) هي الآلية التي تضمن أن مجموعة من العمليات إما تنجح جميعها أو تفشل جميعها (Atomicity). في سياق قاعدة البيانات البيولوجية، هذا يعني أنه عند حفظ جين مع نسخه وطفراته، إما أن يتم حفظ كل شيء بنجاح، أو لا يتم حفظ أي شيء.

Transactions are the mechanism that ensures a set of operations either all succeed or all fail (Atomicity). In the context of a biological database, this means that when saving a gene with its transcripts and mutations, either everything is saved successfully, or nothing is saved at all.

22.1 ACID في قواعد البيانات العلائقية

22.1 ACID in Relational Databases.

قواعد البيانات العلائقية تضمن أربع خصائص أساسية تُعرف بـ ACID:

Relational databases guarantee four fundamental properties known as ACID:

Atomicity الكل أو لا شيء حفظ جين مع 5 نسخ و 12 طفرة: إما أن تنجح جميع الإدراجات، أو لا يتم إدراج أي شيء.

Consistency البيانات تبقى صحيحة لا يمكن إدراج طفرة في موقع غير موجود على التسلسل (بسبب FOREIGN KEY و CHECK).

الخاصية (Property) المعنى (Meaning) المثال في مشروعنا (Example in Our Project)

Isolation المعاملات المتزامنة لا تتداخل عمليتا حفظ لجينين مختلفين لا تؤثر إحداهما على الأخرى.

Durability البيانات محفوظة بعد الـ Commit بعد commit()، تبقى البيانات حتى في حال انقطاع الكهرباء.

Atomicity | All or Nothing | Saving a gene with 5 transcripts and 12 mutations: either all inserts succeed, or nothing is inserted.

Consistency | Data remains valid | Cannot insert a mutation at a position that does not exist on the sequence (due to FOREIGN KEY and CHECK).

Isolation | Concurrent transactions do not interfere | Two separate save operations for different genes do not affect each other.

Durability | Data persists after Commit | After commit(), data remains even in case of power failure.

22.2 تنفيذ المعاملات في sqlite3 مع Python

22.2 Implementing Transactions in sqlite3 with Python.

Python

```
class GeneService:
    """خدمة تتعامل مع منطق الأعمال وتستخدم المستودع."""
```

```
Service handling business logic and using the repository.
"""
def __init__(self, repository: GeneRepositoryInterface):
    self.repository = repository

def save_gene_complete(self, gene: Gene) -> bool:
    """حفظ جين كامل مع معالجة الأخطاء والمعاملات.
    Save a complete gene with error handling and transactions.
    """
    try:
        # المستودع يتعامل مع المعاملة داخليا (في save())
        # The repository handles the transaction internally (in save())
        saved_gene = self.repository.save(gene)
        print(f"✅ تم حفظ الجين: {saved_gene.gene_symbol}")
        return True

    except sqlite3.IntegrityError as e:
        print(f"❌ خطأ في سلامة البيانات: {e}")
        return False

    except sqlite3.OperationalError as e:
        print(f"❌ خطأ تشغيلي في قاعدة البيانات: {e}")
        return False

    except Exception as e:
        print(f"❌ خطأ غير متوقع: {e}")
        return False

def transfer_gene_data(self, source_symbol: str, target_symbol: str) -> bool:
    """نسخ بيانات جين إلى جين آخر (معاملة معقدة).
    Copy data from one gene to another (complex transaction).
    """
```

```
source = self.repository.find_by_symbol(source_symbol)
if not source:
    return False

# إنشاء جين جديد باسم مختلف
new_gene = Gene(
    gene_id=0,  # جديد
    gene_symbol=target_symbol,
    chromosome=source.chromosome,
    strand=source.strand,
    description=f"Copied from {source_symbol}"
)

# نسخ النسخ (نسخ عميق للكائنات)
for transcript in source.transcripts:
    new_transcript = Transcript(
        transcript_id=0,
        gene_id=0,  # سيتم تعيينه عند الحفظ
        transcript_symbol=f"{target_symbol}-001",
        sequence_rna=transcript.sequence_rna,
        length=transcript.length,
        is_coding=transcript.is_coding,
        gc_content=transcript.gc_content,
        organism=transcript.organism
    )
    # نسخ الطفرات
    for mutation in transcript.mutations:
        new_transcript.mutations.append(
            Mutation(
                mutation_id=0,
                position=mutation.position,
                ref_base=mutation.ref_base,
```

```
                alt_base=mutation.alt_base,
                effect=mutation.effect
            )
        )
    new_gene.transcripts.append(new_transcript)

# حفظ الجين الجديد في معاملة واحدة
return self.save_gene_complete(new_gene)
```

الفصل الثالث والعشرون: التحقق من صحة البيانات (**Validation**) — الدفاع عن سلامة المدخلات

Chapter Twenty-Three: Data Validation — Defending Input Integrity

التحقق من صحة البيانات يجب أن يعمل على طبقتين متكاملتين. طبقة التطبيق تستطيع تقديم رسائل مفهومة والتحقق من القواعد التي تعتمد على سياق العمل، بينما يجب أن تحمي قاعدة البيانات القواعد التي لا يجوز كسرها مهما كان مصدر الإدخال. هذا الفصل مهم لأن البيانات العلمية لا تصبح صحيحة لمجرد أن Python قبلتها؛ قد تكون السلسلة صالحة نحويًا لكنها مرتبطة بعينة خاطئة، أو قد يكون تاريخ التجربة غير متسق مع إصدار المرجع. لذلك نحتاج إلى Validation وConstraints وProvenance معًا.

Data validation should operate through two complementary layers. The application layer can provide clear messages and enforce context-dependent business rules, while the database should protect invariants that must never be violated regardless of the input source. This matters because scientific data do not become correct merely because Python accepted them; a sequence may be syntactically valid yet linked to the wrong sample, or an experiment date may be inconsistent with the reference version. Validation, constraints, and provenance therefore have to work together.

مختبر: تقرير تحقق قابل لإعادة الإنتاج

Laboratory: A Reproducible Validation Report

يجب أن يميز التحقق بين الفقد والتكرار والفشل البنيوي وفشل القاعدة البيولوجية. ينبغي عدّ هذه الفئات منفصلة لأنها تقود إلى إجراءات تصحيح مختلفة.

Validation should distinguish missingness, duplication, structural failure, and biological-rule failure. These categories should be counted separately because they imply different corrective actions.

يجب حفظ تقرير التحقق نفسه مع تشغيل التحليل حتى يمكن تفسير النتائج اللاحقة في ضوء جودة المدخلات الدقيقة.

A validation report should itself be saved with the analysis run so that later results can be interpreted in light of the exact input quality.

التحقق كأثر علمي مسجل

Validation as a recorded scientific artifact

```
from dataclasses import dataclass
```

```python
@dataclass(frozen=True)
class ValidationReport:
    total: int
    missing: int
    duplicates: int
    invalid: int


def summarize(records):
    ids=[]; missing=0; invalid=0
    for r in records:
        if not r.get('sequence'):
            missing += 1
        if not r.get('gene_symbol'):
            invalid += 1
        ids.append(r.get('sample_id'))
    duplicates = len(ids) - len(set(ids))
    return ValidationReport(len(records), missing, duplicates, invalid)
```

PYTHON

الشكل 23 —— الخريطة المفاهيمية للفصل

Figure 23 — Conceptual map of the chapter

الشكل 23 —— سير العمل الهندسي للفصل

Figure 23 — Engineering workflow of the chapter

23.5 NULL, None, and Missing Data — Three Concepts That Must Not Be Confused.

23.5 NULL وNone وMissing Data —— ثلاث أفكار لا يجوز خلطها

القيمة المفقودة ليست دائمًا صفرًا، وليست دائمًا سلسلة فارغة، وليست دائمًا False. في NULL ،SQL تعني غياب قيمة مع دلالات منطقية خاصة. في None ،Python كائن يشير إلى عدم وجود قيمة. أما missing data في العلوم فقد تعني أن القياس لم يُجرَ، أو لم يصل، أو لم يكن صالحًا، أو لم يكن منطبقًا.

A missing value is not always zero, an empty string, or False. In SQL, NULL represents the absence of a value with special logical semantics. In Python, None is an object used to represent no value. In science, missing data may mean that a measurement was not performed, was unavailable, was invalid, or was not applicable.

Python

```
def normalize_optional(value):
    if value is None:
        return None
    value = value.strip()
    return value if value else None
```

SQL

```
SELECT
    sample_id,
    CASE
        WHEN collection_date IS NULL THEN 'missing'
```

```
        ELSE 'present'
    END AS date_status
FROM samples;
```

عند بناء التحليل، احتفظ بسبب الغياب عندما يكون مهمًا علميًا. تحويل كل الحالات إلى NULL قد يسهل التخزين، لكنه قد يدمر معنى البيانات.

When building an analysis, preserve the reason for absence whenever it is scientifically meaningful. Converting every case to NULL may simplify storage, but it can destroy the meaning of the data.

الخطوة التالية في هذا المسار تبدأ مما أُنجز في الجزء الثاني، تعلمنا البرمجة الدفاعية (Defensive Programming) والتحقق من المدخلات قبل استخدامها. هذا المبدأ لا يقل أهمية عند التعامل مع قاعدة البيانات. يجب أن نتحقق من صحة البيانات قبل إرسالها إلى SQL، لأن قاعدة البيانات قد ترفضها (بسبب القيود) لكن رسائل الخطأ قد تكون غامضة للمستخدم.

What you learned earlier about data and programming now becomes the starting point, we learned Defensive Programming and validating inputs before use. This principle is no less important when dealing with a database. We must validate data before sending it to SQL, because the database might reject it (due to constraints) but the error messages might be cryptic to the user.

23.1 التحقق من صحة الكائنات قبل الحفظ

23.1 Validating Objects Before Saving.

Python

```
from typing import List, Optional


class ValidationError(Exception):
    """استثناء مخصص لأخطاء التحقق.
    Custom exception for validation errors.
    """
    pass


def validate_gene(gene: Gene) -> List[str]:
    """التحقق من صحة كائن Gene. إرجاع قائمة بالأخطاء إن وجدت.
    Validate a Gene object. Return a list of errors if any.
    """
    errors = []

    if not gene.gene_symbol or len(gene.gene_symbol) < 2:
        errors.append("رمز الجين يجب أن لا يقل عن حرفين")

    if gene.chromosome not in [str(i) for i in range(1, 23)] + ["X", "Y", "MT"]:
        errors.append(f"كروموسوم غير صالح: {gene.chromosome}")

    if gene.strand not in (1, -1):
```

```
        errors.append(f"الاتجاه يجب أن يكون 1 أو -1، وليس {gene.strand}")

    if gene.avg_expression is not None and gene.avg_expression < 0:
        errors.append("متوسط التعبير لا يمكن أن يكون سالبا")

    return errors

def validate_transcript(transcript: Transcript) -> List[str]:
    """التحقق من صحة كائن Transcript.
    Validate a Transcript object.
    """
    errors = []

    if not transcript.transcript_symbol:
        errors.append("رمز النسخة مطلوب")

    if not transcript.sequence_rna or len(transcript.sequence_rna) < 10:
        errors.append("تسلسل الـ RNA يجب أن لا يقل عن 10 قواعد")

    if transcript.gc_content is not None and not (0 <= transcript.gc_content <= 100):
        errors.append("نسبة GC يجب أن تكون بين 0 و 100")

    return errors

def validate_mutation(mutation: Mutation) -> List[str]:
    """التحقق من صحة كائن Mutation.
    Validate a Mutation object.
    """
    errors = []

    if mutation.position < 1:
        errors.append(f"موقع الطفرة يجب أن يكون موجبا: {mutation.position}")
```

```
    valid_bases = {'A', 'T', 'C', 'G', 'U'}
    if mutation.ref_base not in valid_bases:
        errors.append(f"القاعدة المرجعية غير صالحة: {mutation.ref_base}")
    if mutation.alt_base not in valid_bases:
        errors.append(f"القاعدة البديلة غير صالحة: {mutation.alt_base}")

    return errors
```

23.2 دمج التحقق في المستودع والخدمة

23.2 Integrating Validation into Repository and Service.

Python

```
class ValidatingGeneRepository(GeneRepositoryInterface):
    """مستودع يتحقق من صحة البيانات قبل إرسالها إلى قاعدة البيانات.
    Repository that validates data before sending to the database.
    """
    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path
        self._ensure_schema()

    def save(self, gene: Gene) -> Gene:
        # 1. التحقق من صحة الجين
        errors = validate_gene(gene)
        if errors:
            raise ValidationError(f"بيانات الجين غير صالحة: {', '.join(errors)}")
```

```
    # 2. التحقق من صحة النسخ والطفرات
    for transcript in gene.transcripts:
        transcript_errors = validate_transcript(transcript)
        if transcript_errors:
            raise ValidationError(
                f"بيانات النسخة {transcript.transcript_symbol} غير صالحة: {', '.join(transcript_errors)}"
            )

        for mutation in transcript.mutations:
            mutation_errors = validate_mutation(mutation)
            if mutation_errors:
                raise ValidationError(
                    f"بيانات الطفرة في الموقع {mutation.position} غير صالحة: {', '.join(mutation_errors)}"
                )

    # 3. بعد اجتياز التحقق، نمرر إلى التخزين
    # 3. After passing validation, pass to storage
    return self._save_to_db(gene)

def _save_to_db(self, gene: Gene) -> Gene:
    """التنفيذ الفعلي للحفظ (نفس الكود من الفصل السابق).
    Actual save implementation (same code from previous chapter).
    """
    # تم حذف التفاصيل للاختصار
    # Details omitted for brevity
    return gene
```

الفصل الرابع والعشرون: بناء طبقة بيانات قابلة لإعادة الاستخدام — التجميع النهائي

Chapter Twenty-Four: Building a Reusable Data Layer — The Final Assembly

الشكل 24 — الخريطة المفاهيمية للفصل

Figure 24 — Conceptual map of the chapter

عندما نصل إلى هذه الطبقة الهندسية، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

الشكل 24 — سير العمل الهندسي للفصل

Figure 24 — Engineering workflow of the chapter

The next engineering step grows out of the work completed in Part Two, you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

هذا الفصل يجمع كل ما تعلمناه في الباب الخامس في هيكل متكامل وقابل لإعادة الاستخدام. سنبني طبقة بيانات كاملة تتكون من:

2. مستودعات (Repositories) لكل كيان رئيسي.

1. كائنات OOP (@dataclass).

3. خدمات (Services) تحتوي على منطق الأعمال (Business Logic).

4. مدير قاعدة بيانات (Database Manager) يدير الاتصالات والمعاملات.

This chapter brings together everything we learned in Unit Five into an integrated, reusable structure. We will build a complete data layer consisting of:

1. OOP Objects (@dataclass).

2. Repositories for each major entity.

3. Services containing Business Logic.

4. Database Manager handling connections and transactions.

24.1 مدير قاعدة البيانات (DatabaseManager)

24.1 DatabaseManager

Python

```
class DatabaseManager:
    """مدير مركزي للاتصالات بقاعدة البيانات.
    Central manager for database connections.
    """
    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path

    def get_connection(self) -> sqlite3.Connection:
        """إنشاء اتصال جديد بقاعدة البيانات.
        Create a new database connection.
        """
        connection = sqlite3.connect(self.db_path)
        connection.row_factory = sqlite3.Row
        return connection

    def execute_in_transaction(self, operations: List[tuple]) -> bool:
```

```
"""تنفيذ مجموعة من العمليات في معاملة واحدة.
Execute a set of operations in a single transaction.
operations: list of (query, params) tuples.
"""
connection = None
try:
    connection = self.get_connection()
    cursor = connection.cursor()

    for query, params in operations:
        cursor.execute(query, params or ())

    connection.commit()
    return True

except Exception as e:
    if connection:
        connection.rollback()
    print(f"Transaction failed: {e}")
    return False

finally:
    if connection:
        connection.close()
```

24.2 المستودعات المتخصصة

24.2 Specialized Repositories.

Python

```
class PatientRepository:
    def __init__(self, db_manager: DatabaseManager):
        self.db_manager = db_manager

    def save(self, patient: Patient) -> Patient:
        with self.db_manager.get_connection() as conn:
            cursor = conn.cursor()
            if patient.patient_id == 0:
                cursor.execute(
                    "INSERT INTO patients (external_id, age, sex, diagnosis) VALUES (?, ?, ?, ?)",
                    (patient.external_id, patient.age, patient.sex, patient.diagnosis)
                )
                patient.patient_id = cursor.lastrowid
            else:
                cursor.execute(
                    "UPDATE patients SET external_id=?, age=?, sex=?, diagnosis=? WHERE patient_id=?",
                    (patient.external_id, patient.age, patient.sex, patient.diagnosis, patient.patient_id)
                )
            conn.commit()
            return patient

    def find_by_external_id(self, external_id: str) -> Optional[Patient]:
        with self.db_manager.get_connection() as conn:
            cursor = conn.cursor()
            cursor.execute(
                "SELECT * FROM patients WHERE external_id = ?",
                (external_id,)
            )
            row = cursor.fetchone()
```

```
        if not row:
            return None
        return Patient(
            patient_id=row["patient_id"],
            external_id=row["external_id"],
            age=row["age"],
            sex=row["sex"],
            diagnosis=row["diagnosis"]
        )

# مستودعات مماثلة لـ SampleRepository، RunRepository، ResistanceRepository، إلخ.
# Similar repositories for SampleRepository, RunRepository, ResistanceRepository, etc.
```

24.3 الخدمات (Services) — منطق الأعمال

24.3 Services — Business Logic.

Python

```
class GeneResistanceService:
    """خدمة متخصصة في تحليل مقاومة الجينات.
    Specialized service for gene resistance analysis.
    """
    def __init__(
        self,
        gene_repo: GeneRepositoryInterface,
        mutation_repo: MutationRepositoryInterface,
        resistance_repo: ResistanceRepositoryInterface
```

```
):
    self.gene_repo = gene_repo
    self.mutation_repo = mutation_repo
    self.resistance_repo = resistance_repo

def find_resistance_mutations_by_gene(
    self,
    gene_symbol: str,
    min_confidence: float = 0.7
) -> List[MutationResistanceView]:
    """استرجاع الطفرات المرتبطة بالمقاومة لجين معين.
    Retrieve resistance-associated mutations for a specific gene.
    """
    gene = self.gene_repo.find_by_symbol(gene_symbol)
    if not gene:
        return []

    results = []
    for transcript in gene.transcripts:
        for mutation in transcript.mutations:
            # استعلام عن المقاومة المرتبطة بهذه الطفرة
            resistance_links = self.resistance_repo.find_by_mutation(
                mutation.mutation_id,
                min_confidence
            )
            for link in resistance_links:
                results.append(
                    MutationResistanceView(
                        position=mutation.position,
                        ref_base=mutation.ref_base,
                        alt_base=mutation.alt_base,
                        effect=mutation.effect,
```

```
                gene_symbol=gene.gene_symbol,
                tissue_type="unknown",  # سيكون من العينة المرتبطة
                drug_name=link.drug_name,
                confidence=link.confidence
            )
        )
    return results

def get_mutation_statistics_by_gene(self, gene_symbol: str) -> dict:
    """إحصائيات الطفرات لجين معين.
    Mutation statistics for a specific gene.
    """
    gene = self.gene_repo.find_by_symbol(gene_symbol)
    if not gene:
        return {}

    total_mutations = 0
    mutation_types = {"missense": 0, "nonsense": 0, "silent": 0, "frameshift": 0}

    for transcript in gene.transcripts:
        total_mutations += len(transcript.mutations)
        for mutation in transcript.mutations:
            if mutation.effect in mutation_types:
                mutation_types[mutation.effect] += 1

    return {
        "gene_symbol": gene.gene_symbol,
        "total_mutations": total_mutations,
        "mutation_types": mutation_types,
        "transcript_count": len(gene.transcripts)
    }
```

24.4 هيكل المشروع النهائي (Final Project Structure)

24.4 Final Project Structure.

Code

```
genomic_engine/
p    yproject  .toml
├── README.md
├── src/
g   e    nomic_engine/
│    ├── __init__.py
│    ├── models.py       # كائنات OOP (Gene, Transcript, Mutation, إلخ)
│    ├── database.py     # DatabaseManager
│    ├── repositories.py  # جميع المستودعات (Gene, Patient, Sample, إلخ)
│    ├── services.py     # منطق الأعمال (GeneResistanceService, إلخ)
│    ├── validators.py   # دوال التحقق من الصحة
│    └── cli.py          # واجهة سطر الأوامر
├── tests/
│  ├── test_models.py
│  ├── test_repositories.py
│  ├── test_services.py
│  └── test_validators.py
├── data/
│  └── genomic_data.db     # قاعدة البيانات (تنشأ تلقائياً)
└── docs/
└── api.md
```

الانتقال إلى الباب السادس:

Transition to Unit Six:

الآن أصبح لدينا طبقة بيانات متكاملة: OOP + SQL + Repositories + Services + Validation. هذا هو قلب النظام. لكن البيانات ستكبر، والاستعلامات ستصبح أثقل. في الباب السادس، سنتعلم كيف نتعامل مع البيانات الكبيرة (Big Data) في السياق الجينومي: الفهرسة (Indexing)، وتحسين الاستعلامات (Query Optimization)، والمعالجة بالدفعات (Batch Processing)، والتوازي (Parallelism). سننتقل من "قاعدة بيانات تعمل" إلى "قاعدة بيانات تعمل بسرعة وكفاءة على نطاق واسع".

Now we have an integrated data layer: OOP + SQL + Repositories + Services + Validation. This is the heart of the system. However, data will grow, and queries will become heavier. In Unit Six, we will learn how to handle Big Data

in the genomic context: Indexing, Query Optimization, Batch Processing, and Parallelism. We will move from "a working database" to "a fast and efficient database at scale."

—

الباب السادس: من قواعد البيانات إلى البيانات الكبيرة

Unit Six: From Databases to Big Data

الغرض من هذا الباب:

مع نمو البيانات الجينومية، تبدأ قاعدة البيانات المحلية في التباطؤ. الاستعلامات التي كانت سريعة على 1000 تسلسل تصبح بطيئة على 10 ملايين. هذا الباب يعلمك كيف تشخص عنق الزجاجة (Bottleneck)، وكيف تستخدم الفهارس (Indexing) وتحسين الاستعلامات (Query Optimization) لإعادة السرعة، وكيف تتعامل مع المعالجة بالدفعات (Batch Processing) والتوازي (Parallelism) عندما تصبح البيانات أكبر من الذاكرة المتاحة. وأخيراً، متى تدرك أن قاعدة البيانات المحلية لم تعد

كافية، وحان الوقت للانتقال إلى الحوسبة عالية الأداء (HPC) أو الأنظمة الموزعة.

Purpose of this unit:

As genomic data grows, the local database starts to slow down. Queries that were fast on 1,000 sequences become slow on 10 million. This unit teaches you how to diagnose bottlenecks, how to use Indexing and Query Optimization to regain speed, and how to handle Batch Processing and Parallelism when data exceeds available memory. Finally, when to realize that a local database is no longer sufficient, and it is time to move to High-Performance Computing (HPC) or distributed systems.

الفصل الخامس والعشرون: حجم البيانات والتعقيد الحسابي — إعادة النظر في **Big-O** في سياق **SQL**

Chapter Twenty-Five: Data Volume and Computational Complexity — Revisiting Big-O in the SQL Context

الشكل 25 — الخريطة المفاهيمية للفصل

Figure 25 — Conceptual map of the chapter

الشكل 25 — سير العمل الهندسي للفصل

Figure 25 — Engineering workflow of the chapter

في الجزء الثاني (الفصل 26)، تعلمت عن التعقيد الحسابي (Big-O) وكيف يؤثر على خوارزميات المحاذاة والبحث. في عالم قواعد البيانات، نفس المبادئ تنطبق، لكن مع تغيير جوهري: المحرك (Engine) يقوم بالتحسين نيابة عنك، لكنه لا يستطيع

فعل المعجزات. فهم التعقيد الحسابي للاستعلامات يساعدك على توقع الأداء وتصميم الفهارس المناسبة.

In Part Two (Chapter 26), you learned about Computational Complexity (Big-O) and how it affects alignment and search algorithms. In the database world, the same principles apply, but with a fundamental difference: the Engine optimizes on your behalf, but it cannot work miracles. Understanding the computational complexity of queries helps you predict performance and design appropriate indexes.

25.1 تعقيد الاستعلامات بدون فهارس (Full Table Scan)

25.1 Query Complexity Without Indexes (Full Table Scan).

عندما تنفذ استعلاماً بدون فهرس مناسب، يقوم محرك قاعدة البيانات بـ مسح كامل للجدول (Full Table Scan). هذا يعني أنه يقرأ كل صف في الجدول ليجد المطابقات. تعقيده

هو O(N)، حيث N هو عدد الصفوف. إذا كان N = 10 ملايين، والاستعلام يحتوي على JOIN بين جدولين، يصبح التعقيد O(N * M) في أسوأ الحالات.

When you execute a query without a suitable index, the database engine performs a Full Table Scan. This means it reads every row in the table to find matches. Its complexity is O(N), where N is the number of rows. If N = 10 million, and the query contains a JOIN between two tables, the complexity becomes O(N * M) in the worst case.

مثال كارثي (استعلام بدون فهارس):

Catastrophic Example (Query Without Indexes):

SQL

```
-- هذا الاستعلام سيجبر المحرك على مسح جداول ضخمة إذا لم توجد فهارس على patient_id أو sample_id
-- This query will force the engine to scan massive tables if no indexes exist on patient_id or sample_id
SELECT p.external_id, s.tissue_type, seq.gene_symbol
FROM patients p
JOIN samples s ON p.patient_id = s.patient_id
JOIN sequences seq ON s.sample_id = seq.sample_id
```

```
WHERE p.age > 50 AND seq.gene_symbol = 'BRCA1';
```

التعقيد المقدر بدون فهارس:

· مسح patients: N1 (مثلاً 100,000)

· مسح samples: N2 (مثلاً 1,000,000)

· التعقيد التقريبي: O(N1 + N2 + N3) ≈ 11 مليون عملية قراءة. مع JOIN، قد يصبح قريباً من O(N1 * N2) في خوارزميات التجزئة السيئة.

· مسح sequences: N3 (مثلاً 10,000,000)

Estimated Complexity Without Indexes:

· Scan patients: N1 (e.g., 100,000)

· Scan samples: N2 (e.g., 1,000,000)

· Scan sequences: N3 (e.g., 10,000,000)

· Approximate complexity: O(N1 + N2 + N3) ≈ 11 million read operations. With JOIN, it could approach O(N1 * N2) in poor hash algorithms.

الحل: الفهارس تحول التعقيد من O(N) إلى O(log N) أو حتى O(1) في بعض الحالات.

The Solution: Indexes transform complexity from O(N) to O(log N) or even O(1) in some cases.

25.2 ربط التعقيد بتحليل الأداء العملي (EXPLAIN QUERY PLAN)

25.2 Linking Complexity to Practical Performance Analysis (EXPLAIN QUERY PLAN).

في SQLite، يمكنك استخدام EXPLAIN QUERY PLAN لرؤية كيفية تنفيذ الاستعلام فعلياً. هذا يشبه time في Bash لكن على مستوى المنطق.

In SQLite, you can use EXPLAIN QUERY PLAN to see how the query is actually executed. This is like time in Bash but at the logic level.

SQL

```
EXPLAIN QUERY PLAN
SELECT p.external_id, s.tissue_type
FROM patients p
JOIN samples s ON p.patient_id = s.patient_id
WHERE p.age > 50;
```

المخرجات المحتملة (بدون فهارس):

Possible Output (Without Indexes):

Code

```
SCAN patients
SCAN samples
```

هذا يعني O(N * M) أو مسح كامل لكلا الجدولين.

Output After Adding Indexes on age in patients and patient_id in samples:

المخرجات بعد إضافة فهرس على age في patients و patient_id في samples:

Code

```
SEARCH patients USING INDEX idx_patients_age (age>?)
SEARCH samples USING INDEX idx_samples_patient_id (patient_id=?)
```

هنا أصبح التعقيد O(log N) لكل جدول، مما يقلل الوقت من ثوانٍ إلى أجزاء من الثانية.

الفصل السادس والعشرون: الفهرسة (**Indexing**) — التسريع الذكي للاستعلامات

Chapter Twenty-Six: Indexing — Smart Query Acceleration

الفهرس ليس «زر تسريع» مجانيًا. كل فهرس يضيف تكلفة تخزين وتحديث، وقد لا يفيد إذا كان الاستعلام يعيد معظم الصفوف أو إذا كان ترتيب الأعمدة غير مناسب لنمط البحث. لذلك يجب أن يبدأ تصميم الفهرس من الأسئلة العلمية الفعلية: ما الذي تبحث عنه كثيرًا؟ ما الذي يفرّق بين عدد كبير من الصفوف؟ وما الذي يحتاج إلى ترتيب؟ ثم يُختبر أثر الفهرس بخطة التنفيذ وقياس فعلي، لا بالحدس وحده.

An index is not a free “speed button.” Every index adds storage and update cost, and an index may provide little benefit when a query returns most rows or when the column order does not match the access pattern. Index design should therefore begin with the scientific questions you actually ask: what is searched frequently, what sharply reduces the candidate rows, and what needs ordering? The effect should then be evaluated with an execution plan and measurements rather than intuition alone.

الشكل 26 — الخريطة المفاهيمية للفصل

Figure 26 — Conceptual map of the chapter

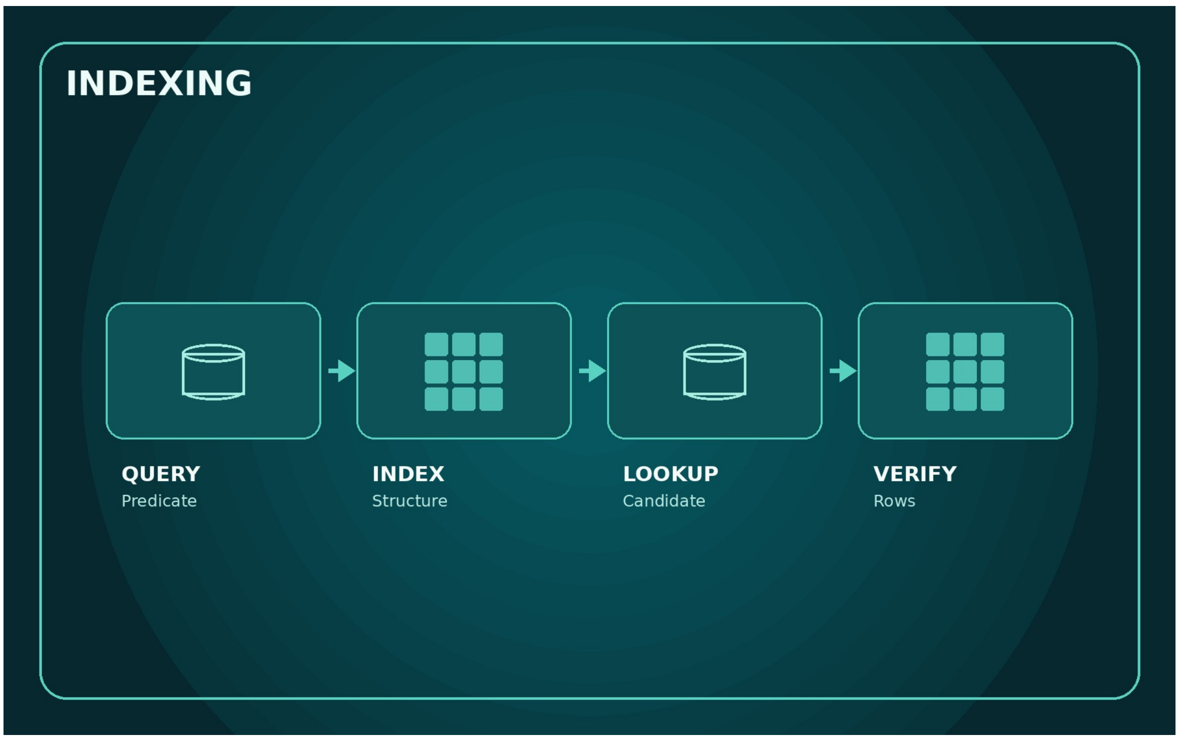
الشكل 26 — سير العمل الهندسي للفصل

Figure 26 — Engineering workflow of the chapter

الخبرة السابقة في Python وBiopython تمهّد مباشرة لهذه الخطوة، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

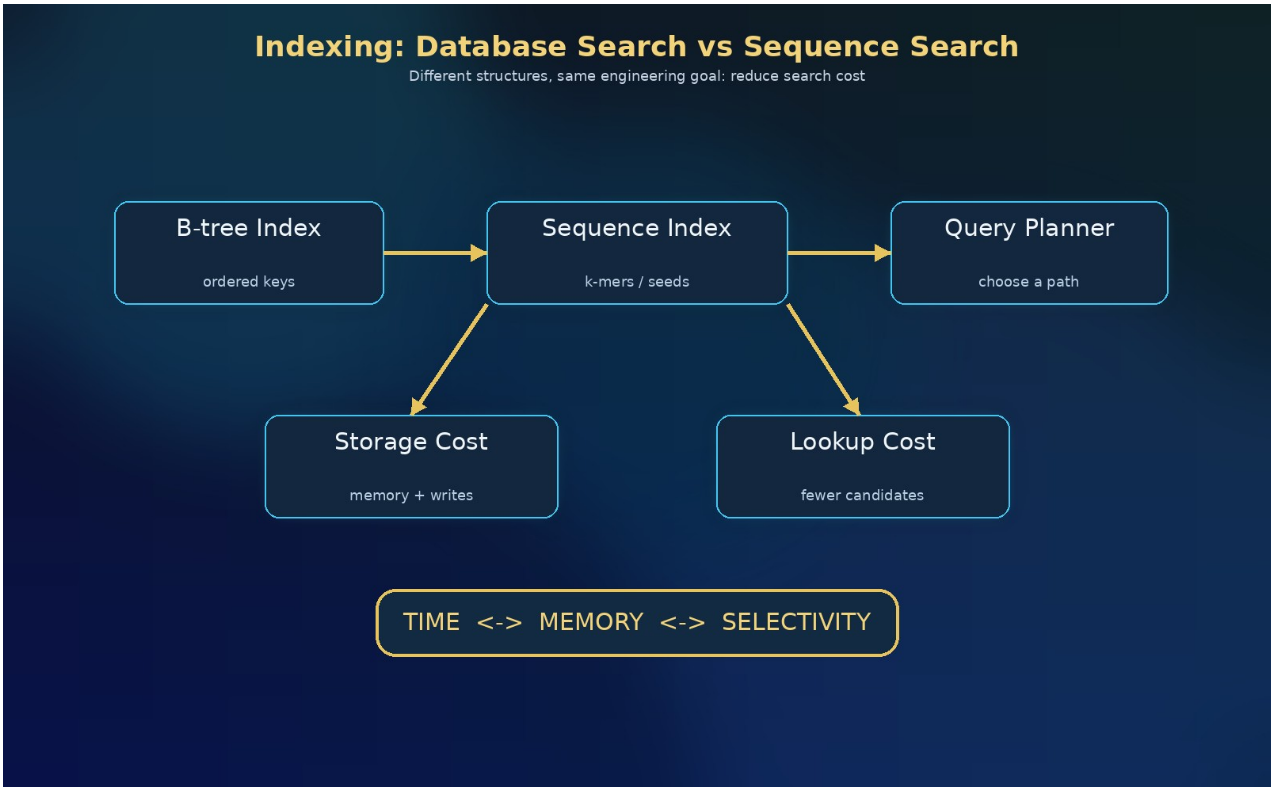
When we reach this engineering layer, you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

26.5 SQL Indexing versus Sequence Indexing — The Shared Engineering Principle.



26.5 فهرسة SQL مقابل فهرسة التسلسلات — المبدأ الهندسي المشترك

Figure — INDEXING



الفهرسة تقلل العمل غير الضروري؛ والاختيار الصحيح يعتمد على نمط السؤال وانتقائية المفتاح.

الفهرس ليس سحرًا؛ إنه بنية بيانات تدفع تكلفة في التخزين والتحديث كي تقلل تكلفة البحث. في قواعد البيانات قد تستخدم B-tree أو GIN وفق طبيعة البيانات والاستعلام. وفي BLAST توجد آليات فهرسة للتسلسلات النووية تعتمد على مواقع N-mer لتجنب المسح الكامل في ظروف مناسبة. توثيق

NCBI يوضح أن الفهرسة قد تحسن زمن البحث لكنها تضيف تكلفة تخزين وقد تكون غير مفيدة إذا لم يتسع الفهرس في الذاكرة.

Indexing reduces unnecessary work; the right structure depends on the query pattern and key selectivity.

An index is not magic; it is a data structure that spends storage and update cost to reduce search cost. Databases may use B-tree or GIN according to the data and query pattern. BLAST also provides indexed nucleotide searches using pre-indexed N-mer locations so that the whole database need not be scanned under suitable conditions. NCBI documents that indexing can improve search time but adds storage cost and may lose its advantage when the index cannot fit in memory.

SQL

```sql
CREATE INDEX idx_sequences_accession
ON sequences(accession);

CREATE INDEX idx_mutations_gene_position
ON mutations(gene_id, position);
```

الفكرة التي ينبغي أن تثبت في ذهنك هي المقايضة: سرعة الوصول مقابل حجم الفهرس وكلفة الكتابة. لا تنقل بنية

فهرسة BLAST إلى SQL حرفيًا، ولا تفترض أن كل فهرس يصلح لكل استعلام.

The principle to retain is the trade-off: access speed versus index size and write cost. Do not copy a BLAST index structure literally into SQL, and do not assume that every index is appropriate for every query.

الفهرس (Index) هو بنية بيانات مساعدة (غالباً شجرة B-tree) تسمح لمحرك قاعدة البيانات بالعثور على الصفوف بسرعة دون مسح الجدول بأكمله. لكن الفهرسة لها ثمن: تستهلك مساحة تخزين إضافية، وتبطئ عمليات الإدراج (INSERT) والتحديث (UPDATE) لأن الفهرس يجب أن يُحدَّث أيضاً.

An Index is an auxiliary data structure (often a B-tree) that allows the database engine to find rows quickly without scanning the entire table. However, indexing has a cost: it consumes additional storage space, and slows down INSERT and UPDATE operations because the index must also be updated.

26.1 أنواع الفهارس في SQLite

26.1 Index Types in SQLite

نوع الفهرس (Index Type) الوصف (Description) الاستخدام الأمثل (Best Use)

B-tree (افتراضي) هيكل شجري متوازن، بحث O(log N). معظم الاستعلامات، خاصة =، >، <، BETWEEN، و LIKE (بدون بداية %).

UNIQUE INDEX B-tree مع قيد تفرد. الأعمدة التي يجب أن تكون فريدة (مثل external_id).

Composite Index فهرس على عدة أعمدة. الاستعلامات التي تفلتر حسب عدة أعمدة معاً (مثل (gene_symbol, chromosome)).

Partial Index فهرس على جزء من الصفوف (شرط WHERE). عندما تبحث فقط على مجموعة فرعية (مثل الطفرات ذات التأثير الخطير فقط).

Full-Text Search (FTS) فهرس خاص للنصوص الطويلة. البحث في تسلسلات DNA أو البروتين الطويلة (بديل لـ LIKE '%...%').

مثال: فهرس مركب (Composite Index) لتسريع استعلامات الجينات حسب الرمز والكروموسوم:

Example: Composite Index to Speed Up Gene Queries by Symbol and Chromosome:

SQL

```
-- فهرس مركب: البحث عن جينات معينة على كروموسوم محدد
-- Composite Index: Searching for specific genes on a specific chromosome
CREATE INDEX idx_genes_symbol_chromosome ON genes(gene_symbol, chromosome);

-- الآن هذا الاستعلام يصبح سريعاً جدا:
-- Now this query becomes very fast:
SELECT * FROM genes WHERE gene_symbol = 'BRCA1' AND chromosome = '17';
```

مثال: فهرس جزئي (Partial Index) لتسريع البحث عن الطفرات الخطيرة:

Example: Partial Index to Speed Up Searching for Dangerous Mutations:

SQL

```
-- فهرس فقط على الطفرات من نوع nonsense و frameshift (الأكثر تأثيرا)
-- Index only on nonsense and frameshift mutations (the most impactful)
CREATE INDEX idx_mutations_serious ON mutations(effect)
WHERE effect IN ('nonsense', 'frameshift');

-- هذا الاستعلام سيستخدم الفهرس الجزئي بسرعة:
-- This query will use the partial index quickly:
SELECT * FROM mutations WHERE effect = 'nonsense';
```

26.2 ربط فهرسة SQL بفهرسة BLAST (جسر معرفي من الجزء الثاني)

26.2 Linking SQL Indexing to BLAST Indexing (Conceptual Bridge from Part Two).

الانتقال إلى هذا المفهوم يبدأ من مهارة سابقة لديك، تعلمت أن BLAST يستخدم فهرساً قائماً على الكلمات (Word-based Index) لتسريع البحث عن التسلسلات المتشابهة. بدلاً من مقارنة كل تسلسل استعلام مع كل تسلسل في قاعدة البيانات (O(N*M))، يقوم BLAST ببناء جدول تجزئة (Hash Table) لجميع الكلمات (k-mers) بطول معين، مما يقلل البحث إلى O(L) حيث L هو طول الاستعلام.

Your earlier experience with Python and Biopython prepares you directly for this step, you learned that BLAST uses a Word-based Index to speed up searching for similar sequences. Instead of comparing every query sequence with every database sequence (O(N*M)), BLAST builds a Hash Table of all words (k-mers) of a certain length, reducing the search to O(L) where L is the query length.

وجه الشبه مع فهارس SQL:

- B-tree في SQL يشبه شجرة اللاحقات (Suffix Tree) في البحث النصي، حيث يسمح بالبحث السريع عن الأنماط.

· فهرس التجزئة (Hash Index) في SQL (المتوفر في PostgreSQL) يشبه مباشرة جدول التجزئة في BLAST، لكن SQLite لا يدعمه افتراضياً (يستخدم B-tree).

· الفهارس الجزئية في SQL تشبه تصفية BLAST حيث يتم تجاهل التسلسلات المتكررة أو منخفضة التعقيد.

الدرس الهندسي: الفهرسة هي مقايضة كلاسيكية (Time-Space Tradeoff). BLAST يضحي بالذاكرة (يخزن جدول الكلمات) لتوفير الوقت. SQL تفعل الشيء نفسه مع B-tree.

The Engineering Lesson: Indexing is a classic Time-Space Tradeoff. BLAST sacrifices memory (stores the word table) to save time. SQL does the same with B-tree.

الفصل السابع والعشرون: تحسين الاستعلامات (**Query Optimization**) — كتابة استعلامات سريعة

Chapter Twenty-Seven: Query Optimization — Writing Fast Queries

الشكل 27 — الخريطة المفاهيمية للفصل

Figure 27 — Conceptual map of the chapter

الآن نعيد استخدام ما سبق، ولكن في سياق هندسي أوسع، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنتقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

الشكل 27 — سير العمل الهندسي للفصل

Figure 27 — Engineering workflow of the chapter

This concept does not begin from zero; it begins with a skill you already have, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

الفهارس وحدها ليست كافية. ترتيب الشروط، واختيار JOIN المناسب، وتجنب العمليات غير الضرورية، كلها تؤثر على الأداء. المحرك يقوم ببعض التحسين تلقائياً، لكن العادات البرمجية الجيدة تجعله يعمل بكفاءة أعلى.

Indexes alone are not enough. The order of conditions, choosing the right JOIN, and avoiding unnecessary operations all affect performance. The engine does some optimization automatically, but good programming habits make it work even more efficiently.

27.1 قواعد ذهبية لتحسين استعلامات SQL

27.1 Golden Rules for Optimizing SQL Queries.

جلب أعمدة غير ضرورية يزيد من نقل البيانات (I/O) ويستهلك الذاكرة.

1. استخدم SELECT محدداً، وليس * (Use Specific SELECT, not *):

Fetching unnecessary columns increases I/O and consumes memory.

SQL

```
-- سيئ: يجلب كل شيء
-- Bad: Fetches everything
SELECT * FROM transcripts WHERE gene_id = 5;

-- جيد: يجلب فقط ما نحتاج
-- Good: Fetches only what we need
SELECT transcript_id, transcript_symbol, sequence_rna FROM transcripts WHERE gene_id = 5;
```

2. ضع الشروط الأكثر تحديداً أولاً (في WHERE):

المحرك يطبق الشروط بترتيب معين (غالباً حسب وجود الفهارس). ضع الشروط التي تفلتر أكبر عدد من الصفوف في البداية.

Place the most selective conditions first (in WHERE). The engine applies conditions in a certain order (often based on index availability). Place conditions that filter the most rows first.

SQL

```
-- أفضل: gene_symbol يفلتر أكثر من chromosome
-- Better: gene_symbol filters more than chromosome
SELECT * FROM genes WHERE gene_symbol = 'BRCA1' AND chromosome = '17';

-- أسوأ: chromosome يفلتر أقل، ثم gene_symbol
-- Worse: chromosome filters less, then gene_symbol
SELECT * FROM genes WHERE chromosome = '17' AND gene_symbol = 'BRCA1';
```

(ملاحظة: معظم المحركات الحديثة تعيد ترتيب الشروط تلقائياً، لكن من الجيد كتابتها بترتيب منطقي لفهم خطة التنفيذ).

(Note: Most modern engines reorder conditions automatically, but it is good practice to write them in a logical order to understand the execution plan.).

3. تجنب SELECT DISTINCT ما لم يكن ضرورياً:

DISTINCT يتطلب فرزاً أو تجزئة، وهو مكلف. استخدمه فقط عندما تعلم أن هناك تكراراً حقيقياً.

Avoid SELECT DISTINCT unless necessary. DISTINCT requires sorting or hashing, which is expensive. Use it only when you know there is real duplication.

4. استخدم EXISTS بدلاً من IN للقوائم الكبيرة:

EXISTS يتوقف عند أول تطابق، بينما IN قد يجمع قائمة كاملة.

Use EXISTS instead of IN for large subqueries. EXISTS stops at the first match, while IN might collect a full list.

SQL

```sql
-- جيد (يتوقف مبكراً)
-- Good (stops early)
SELECT * FROM patients p
WHERE EXISTS (
   SELECT 1 FROM samples s WHERE s.patient_id = p.patient_id AND s.rna_quality > 8
);

-- قد يكون بطيئاً مع قوائم كبيرة
-- May be slow with large lists
SELECT * FROM patients
WHERE patient_id IN (
   SELECT patient_id FROM samples WHERE rna_quality > 8
);
```

5. تجنب العمليات على الأعمدة المفهرسة في WHERE:

استخدام دوال على عمود مفهرس يمنع استخدام الفهرس.

Avoid operations on indexed columns in WHERE. Using functions on an indexed column prevents index usage.

SQL

```
-- سيئ: يمنع استخدام الفهرس على created_at
-- Bad: Prevents index usage on created_at
SELECT * FROM samples WHERE DATE(collection_date) = '2025-01-01';

-- جيد: يحافظ على الفهرس
-- Good: Preserves the index
SELECT * FROM samples WHERE collection_date >= '2025-01-01' AND collection_date < '2025-01-02';
```

27.2 استخدام EXPLAIN QUERY PLAN لتشخيص البطء

27.2 Using EXPLAIN QUERY PLAN to Diagnose Slowness.

Python

```
def diagnose_query(db_path: str, query: str, params: tuple = ()):
```

```
"""تحليل خطة تنفيذ استعلام وعرضها بطريقة مفهومة.
Analyze query execution plan and display it in an understandable way.
"""
with sqlite3.connect(db_path) as conn:
    cursor = conn.cursor()
    # شرح خطة التنفيذ
    cursor.execute(f"EXPLAIN QUERY PLAN {query}", params)
    plan = cursor.fetchall()

    print("=" * 60)
    print("📊 خطة تنفيذ الاستعلام (Query Execution Plan):")
    for row in plan:
        indent = "  " * row[0]
        print(f"{indent}└─ {row[1]}: {row[2]} (detail: {row[3]})")
    print("=" * 60)

    # تنفيذ الاستعلام الفعلي وقياس الوقت
    import time
    start = time.time()
    cursor.execute(query, params)
    results = cursor.fetchall()
    elapsed = time.time() - start
    print(f"⏱️ وقت التنفيذ: {elapsed:.4f} ثانية")
    print(f"📄 عدد الصفوف المسترجعة: {len(results)}")
    return results
```

الفصل الثامن والعشرون: المعالجة بالدفعات (**Batch Processing**) — إدراج ملايين الصفوف بكفاءة

Chapter Twenty-Eight: Batch Processing — Inserting Millions of Rows Efficiently

الشكل 28 — الخريطة المفاهيمية للفصل

Figure 28 — Conceptual map of the chapter

المفاهيم التي بُنيت في الجزء الثاني تأخذ هنا وظيفة جديدة، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

الشكل 28 — سير العمل الهندسي للفصل

Figure 28 — Engineering workflow of the chapter

We now reuse an earlier idea, but in a broader engineering context, you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

في المعلوماتية الحيوية، غالباً ما تحتاج إلى إدراج آلاف أو ملايين الصفوف دفعة واحدة (مثل نتائج المحاذاة، أو قراءات

التعبير الجيني). الإدراج صفاً صفاً (INSERT واحد لكل صف) بطيء جداً بسبب عدد عمليات الـ Commit والتحقق من القيود. الحل هو الإدراج المجمع (Batch Insert).

In bioinformatics, you often need to insert thousands or millions of rows at once (e.g., alignment results, gene expression reads). Inserting row by row (one INSERT per row) is very slow due to the number of Commit operations and constraint checks. The solution is Batch Insert.

28.1 استخدام executemany للإدراج المجمع

28.1 Using executemany for Batch Insert.

Python

```
def batch_insert_read_counts(db_path: str, read_counts: List[tuple]) -> int:
    """
    إدراج آلاف القراءات دفعة واحدة باستخدام executemany.
    Insert thousands of reads at once using executemany.
    read_counts: list of (sample_id, gene_symbol, expression_count)
    """
```

```
    if not read_counts:
        return 0

    query = """
        INSERT INTO read_counts (sample_id, gene_symbol, expression_count)
        VALUES (?, ?, ?)
    """

    with sqlite3.connect(db_path) as connection:
        cursor = connection.cursor()
        # تنفيذ الإدراج المجمع
        cursor.executemany(query, read_counts)
        connection.commit()
        return len(read_counts)

# مثال: إدراج 100,000 قراءة من ملف CSV
# Example: Inserting 100,000 reads from a CSV file
import csv

def import_reads_from_csv(csv_path: str, db_path: str) -> int:
    batch = []
    batch_size = 10000  # إدراج 10,000 صف في كل دفعة لتجنب استهلاك الذاكرة
    total_inserted = 0

    with open(csv_path, 'r') as f:
        reader = csv.reader(f)
        next(reader)  # تخطي الرأس (header)

        for row in reader:
            sample_id, gene_symbol, expression = int(row[0]), row[1], float(row[2])
            batch.append((sample_id, gene_symbol, expression))
```

```
        if len(batch) >= batch_size:
            total_inserted += batch_insert_read_counts(db_path, batch)
            batch = []

    # إدراج ما تبقى
    if batch:
        total_inserted += batch_insert_read_counts(db_path, batch)

    return total_inserted
```

28.2 استخدام المعاملات (Transactions) لتسريع الإدراج

28.2 Using Transactions to Speed Up Insertion.

في بعض الأحيان، نريد التحكم في حجم المعاملة يدوياً. المعاملة الكبيرة (مليون صف) قد تستهلك ذاكرة مؤقتة كبيرة. المعاملة الصغيرة جداً (100 صف) تزيد من عدد عمليات الـ Commit. الحجم الأمثل يعتمد على النظام، لكن 10,000-50,000 صف لكل معاملة يعتبر نقطة بداية جيدة.

Sometimes, we want to control transaction size manually. A large transaction (1 million rows) may consume large temporary memory. A very small transaction (100 rows) increases the number of Commits. The optimal size depends on the system, but 10,000-50,000 rows per transaction is a good starting point.

Python

```
def batch_insert_with_transaction_control(db_path: str, data: List[tuple], batch_size: int = 10000):
    """إدراج بيانات مع التحكم في حجم المعاملة.
    Insert data with transaction size control.
    """
    with sqlite3.connect(db_path) as connection:
        cursor = connection.cursor()
        total = len(data)
        inserted = 0

        for i in range(0, total, batch_size):
            batch = data[i:i+batch_size]
            cursor.executemany(
                "INSERT INTO read_counts (sample_id, gene_symbol, expression_count) VALUES 
(?, ?, ?)",
                batch
            )
            connection.commit()  # حفظ كل دفعة كمعاملة منفصلة
            inserted += len(batch)
            print(f"تم إدراج {inserted}/{total} صف")

        return inserted
```

الفصل التاسع والعشرون: التوازي (**Parallelism**) — معالجة متعددة الخيوط والعمليات

Chapter Twenty-Nine: Parallelism — Multi-threading and Multi-processing

الشكل 29 — الخريطة المفاهيمية للفصل

Figure 29 — Conceptual map of the chapter

من منظور المسار الكامل للكتاب، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

الشكل 29 — سير العمل الهندسي للفصل

Figure 29 — Engineering workflow of the chapter

The concepts developed in Part Two take on a new role here, you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

عندما تصبح البيانات ضخمة جداً بحيث لا يمكن لمعالج واحد (Core) معالجتها في وقت معقول، نحتاج إلى تقسيم العمل

على عدة معالجات. هناك طريقتان رئيسيتان في Python:
متعدد الخيوط (Threading) المناسب لعمليات I/O (مثل قراءة الملفات)، ومتعدد العمليات (Multiprocessing) المناسب لعمليات CPU المكثفة (مثل الحسابات).

When data becomes so large that a single CPU core cannot process it in a reasonable time, we need to split the work across multiple cores. There are two main approaches in Python: Threading (suitable for I/O-bound tasks like file reading) and Multiprocessing (suitable for CPU-intensive tasks like calculations).

29.1 تحذير هام: SQLite والكتابة المتزامنة

29.1 Important Warning: SQLite and Concurrent Writes.

SQLite ليس مصمماً للكتابة المتزامنة من عدة خيوط/عمليات. دعمه للكتابة المتزامنة محدود (يستخدم قفلاً على مستوى الملف). إذا كنت بحاجة إلى كتابة متزامنة، استخدم PostgreSQL أو غيره. لكن يمكننا استخدام التوازي لـ قراءة البيانات أو لمعالجة ملفات CSV قبل الإدراج.

SQLite is not designed for concurrent writes from multiple threads/processes. Its support for concurrent writes is limited (uses a file-level lock). If you need concurrent writes, use PostgreSQL or others. However, we can use parallelism for reading data or processing CSV files before insertion.

Enabling WAL (Write-Ahead Logging) mode in SQLite to improve concurrency:

تفعيل وضع WAL (Write-Ahead Logging) في SQLite لتحسين التزامن:

Python

```
def enable_wal(db_path: str):
    with sqlite3.connect(db_path) as conn:
        conn.execute("PRAGMA journal_mode=WAL;")
        conn.execute("PRAGMA synchronous=NORMAL;")
```

29.2 معالجة متعددة الخيوط (Threading) لقراءة ملفات متعددة

29.2 Multi-threading (Threading) for Reading Multiple Files.

Python

```
import threading
from concurrent.futures import ThreadPoolExecutor, as_completed

def process_fastq_file(file_path: str, db_path: str) -> int:
    """قراءة ملف FASTQ وحساب عدد القراءات وتحديث قاعدة البيانات.
    Read a FASTQ file, count reads, and update database.
    """
    from Bio import SeqIO
    count = 0
    with open(file_path, 'rt') as handle:
        for _ in SeqIO.parse(handle, "fastq"):
            count += 1
```

```
    # تحديث قاعدة البيانات (عملية كتابة واحدة في النهاية)
    with sqlite3.connect(db_path) as conn:
        conn.execute(
            "UPDATE runs SET total_reads = ? WHERE fastq_path = ?",
            (count, file_path)
        )
        conn.commit()
    return count

def process_files_parallel(file_paths: List[str], db_path: str, max_workers: int = 4):
    """معالجة ملفات FASTQ متعددة بالتوازي باستخدام ThreadPoolExecutor.
    Process multiple FASTQ files in parallel using ThreadPoolExecutor.
    """
    results = {}
    with ThreadPoolExecutor(max_workers=max_workers) as executor:
        future_to_file = {
            executor.submit(process_fastq_file, f, db_path): f
            for f in file_paths
        }

        for future in as_completed(future_to_file):
            file_path = future_to_file[future]
            try:
                count = future.result()
                results[file_path] = count
                print(f"✅ {file_path}: {count} قراءة")
            except Exception as e:
                print(f"❌ فشل في {file_path}: {e}")

    return results
```

29.3 معالجة متعددة العمليات (Multiprocessing) للحسابات الثقيلة

29.3 Multi-processing (Multiprocessing) for Heavy Computations.

في الحسابات الثقيلة (مثل حساب مصفوفات التشابه أو تحليل التعبير)، نستخدم multiprocessing للاستفادة من جميع نوى المعالج.

For heavy computations (e.g., calculating similarity matrices or expression analysis), we use multiprocessing to utilize all CPU cores.

Python

```
from multiprocessing import Pool, cpu_count
import numpy as np


def compute_gc_for_batch(sequence_batch: List[str]) -> List[float]:
    """حساب نسبة GC لمجموعة من التسلسلات (عملية CPU مكثفة).
    Compute GC percentage for a batch of sequences (CPU-intensive).
    """
```

```
    results = []
    for seq in sequence_batch:
        seq_upper = seq.upper()
        gc = (seq_upper.count('G') + seq_upper.count('C')) / len(seq_upper) * 100 if seq else 0
        results.append(gc)
    return results

def compute_gc_parallel(sequences: List[str], batch_size: int = 1000) -> List[float]:
    """حساب GC للعديد من التسلسلات باستخدام معالجة متوازية.
    Compute GC for many sequences using parallel processing.
    """
    # تقسيم التسلسلات إلى دفعات
    batches = [sequences[i:i+batch_size] for i in range(0, len(sequences), batch_size)]

    with Pool(processes=cpu_count()) as pool:
        results_batches = pool.map(compute_gc_for_batch, batches)

    # دمج النتائج
    all_results = []
    for batch_result in results_batches:
        all_results.extend(batch_result)
    return all_results

# مثال: حساب GC لـ 100,000 تسلسل باستخدام جميع النوى
# Example: Compute GC for 100,000 sequences using all cores
# gc_values = compute_gc_parallel(sequences)
```

الفصل الثلاثون: متى تصبح قاعدة البيانات المحلية غير كافية؟

Chapter Thirty: When Does the Local Database Become Insufficient؟

الشكل 30 — الخريطة المفاهيمية للفصل

Figure 30 — Conceptual map of the chapter

هذه الخطوة لا تبدأ من الصفر؛ بل تعتمد على ما سبق بناؤه، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

الشكل 30 — سير العمل الهندسي للفصل

Figure 30 — Engineering workflow of the chapter

From the perspective of the book's complete progression, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

في هذا الفصل، نضع علامات الحدود التي تخبرنا بأن SQLite (أو أي قاعدة بيانات محلية) لم تعد الخيار المناسب، وحان الوقت للانتقال إلى حلول أكثر قوة: خوادم قواعد البيانات (PostgreSQL)، أو الحوسبة عالية الأداء (HPC)، أو الأنظمة الموزعة (Spark، Dask).

In this chapter, we set the boundary markers that tell us SQLite (or any local database) is no longer the right choice, and it is time to move to more powerful solutions: Database Servers (PostgreSQL), High-Performance Computing (HPC), or Distributed Systems (Spark, Dask).

30.1 علامات الخطر (Warning Signs)

30.1 Warning Signs

زمن الاستعلام > 10 ثوانٍ استعلامات JOIN المعقدة تصبح بطيئة حتى مع الفهارس. تحسين الفهارس، إعادة كتابة الاستعلام، أو الانتقال إلى PostgreSQL (يدعم خطط تنفيذ أكثر تقدماً).

حجم قاعدة البيانات > 100 جيجابايت SQLite يدعم قواعد كبيرة لكن الأداء يتراجع، والنسخ الاحتياطي يصبح صعباً. الانتقال إلى PostgreSQL أو MySQL مع خيارات التوزيع.

العلامة (Sign) الوصف (Description) الحل المقترح (Suggested Solution)

كتابة متزامنة من عدة مستخدمين SQLite لا يدعم الكتابة المتزامنة بشكل جيد (قفل الملف). الانتقال إلى PostgreSQL (يدعم MVCC).

تحليل بيانات ضخمة (TB) البيانات لا تتناسب مع قرص واحد، أو تحتاج إلى معالجة موزعة. استخدام Spark أو Dask مع تخزين موزع (S3، HDFS).

الحاجة إلى تكامل مع أدوات خارجية أدوات مثل Tableau أو PowerBI تتعامل بشكل أفضل مع PostgreSQL. استخدام PostgreSQL.

الخلاصة الهندسية:

Engineering Conclusion:

SQLite هو خيار ممتاز للتعليم، والتطوير، والمشاريع الصغيرة إلى المتوسطة (حتى 10-20 جيجابايت مع أداء جيد). لكن عندما يتجاوز المشروع هذه الحدود، يصبح من الهندسة الجيدة التخطيط للترحيل إلى PostgreSQL أو نظام موزع. المخطط (Schema) الذي صممناه في الباب الرابع متوافق مع PostgreSQL (مع تغييرات طفيفة في أنواع البيانات). هذا يعني أن الاستثمار في التصميم الجيد للمخطط يحميك في المستقبل.

SQLite is an excellent choice for education, development, and small-to-medium projects (up to 10-20 GB with good performance). However, when the project exceeds these limits, it becomes good engineering practice to plan for migration to PostgreSQL or a distributed system. The Schema we designed in Unit Four is compatible with PostgreSQL (with minor data type

changes). This means that investing in good Schema design protects you in the future.

30.2 نظرة على PostgreSQL (للتمهيد)

30.2 A Glimpse at PostgreSQL (For Preparation).

الانتقال إلى PostgreSQL يتطلب تغييرات بسيطة في الكود:

- تغيير أنواع البيانات: TEXT يبقى كما هو، INTEGER يبقى، REAL يبقى.
- الاستفادة من ميزات إضافية مثل فهارس GIN للبحث في JSON، و Full-Text Search المتقدم.
- استبدال sqlite3 بـ psycopg2.
- استخدام SERIAL بدلاً من AUTOINCREMENT.

Transitioning to PostgreSQL requires minor code changes:

· Replace sqlite3 with psycopg2.

· Change data types: TEXT remains, INTEGER remains, REAL remains.

· Use SERIAL instead of AUTOINCREMENT.

· Leverage additional features like GIN indexes for JSON search, and advanced Full-Text Search.

Python

```
# مثال على اتصال PostgreSQL (رمز تحضيري، ليس للتشغيل)
# Example PostgreSQL connection (preparatory code, not for execution)
# import psycopg2
# conn = psycopg2.connect(
#     host="localhost",
#     database="genomic_db",
#     user="bio_user",
#     password="secure_password"
# )
# cursor = conn.cursor()
# cursor.execute("SELECT * FROM genes WHERE gene_symbol = %s", ("BRCA1",))
```

التحضير للمستقبل: في المشاريع الهندسية الكبيرة، لا تنتظر حتى تصل إلى حد الألم. خطط للترحيل مسبقاً عن طريق

كتابة كود Python يعتمد على واجهة (Interface) مجردة (كما فعلنا مع GeneRepositoryInterface)، بحيث يمكنك تبديل التطبيق من SQLite إلى PostgreSQL عن طريق تغيير فئة المستودع فقط، دون لمس بقية التطبيق.

Preparation for the Future: In large engineering projects, do not wait until you hit the pain threshold. Plan for migration in advance by writing Python code that depends on an abstract Interface (as we did with GeneRepositoryInterface), so you can switch the implementation from SQLite to PostgreSQL by simply changing the repository class, without touching the rest of the application.

الانتقال إلى الباب السابع:

Transition to Unit Seven:

الآن أصبح لديك نظام قادر على تخزين واستعلام وتحليل البيانات الجينومية على نطاق واسع. لكن تحليل البيانات الجينومية لا يتوقف عند SELECT و JOIN. الأسئلة العلمية الأعمق تتطلب الرياضيات. في الباب السابع، ننتقل من "استرجاع البيانات" إلى "نمذجة البيانات". سنتعلم لماذا نحتاج إلى المتجهات (Vectors) والمصفوفات (Matrices) لتمثيل التعبير الجيني، وكيف يحولنا الجبر الخطي من الإحصاء الوصفي إلى النماذج التنبؤية. هذا الباب هو الجسر الذي سيوصلنا في النهاية إلى الحوسبة الكمومية (Quantum Computing) في الأبواب المتقدمة.

Now you have a system capable of storing, querying, and analyzing genomic data at scale. However, genomic data analysis does not stop at SELECT and JOIN. Deeper scientific questions require Mathematics. In Unit Seven, we move from "data retrieval" to "data modeling". We will learn why we need

Vectors and Matrices to represent gene expression, and how Linear Algebra takes us from descriptive statistics to predictive models. This unit is the bridge that will eventually lead us to Quantum Computing in the advanced units.

الباب السابع: من البيانات إلى الرياضيات — لماذا نحتاج إلى نمذجة رياضية؟

Unit Seven: From Data to Mathematics — Why Do We Need Mathematical Modeling؟

الغرض من هذا الباب:

تحويل التفكير من "كيف أستعلم عن البيانات؟" إلى "كيف أمثل الظاهرة البيولوجية كنظام رياضي؟". في الأبواب السابقة، تعلمت كيفية تخزين البيانات واسترجاعها باستخدام SQL و OOP. لكن الأسئلة العلمية الحقيقية لا تتوقف عند SELECT. الأسئلة مثل "ما مدى تشابه تعبير جين X مع جين Y عبر العينات؟"، أو "كيف يمكننا التنبؤ بمقاومة المضادات الحيوية بناءً على ملف طفري؟" تتطلب تمثيلاً رياضياً للبيانات. هذا الباب يبني الجسر من الجداول والصفوف إلى المتجهات والمصفوفات والدوال، مع إعادة النظر في الإحصاء

والاحتمالات من منظور رياضي، تمهيداً للجبر الخطي الذي سيقودنا في النهاية إلى الحوسبة الكمومية.

Purpose of this unit:

To shift thinking from "how do I query data?" to "how do I represent a biological phenomenon as a mathematical system?". In previous units, you learned how to store and retrieve data using SQL and OOP. However, real scientific questions do not stop at SELECT. Questions like "how similar is the expression of Gene X to Gene Y across samples?", or "how can we predict antibiotic resistance based on a mutational profile?" require a mathematical representation of the data. This unit builds the bridge from tables and rows to vectors, matrices, and functions, revisiting statistics and probability from a mathematical perspective, paving the way for Linear Algebra which will eventually lead us to Quantum Computing.

الفصل الحادي والثلاثون: لماذا نحتاج الرياضيات؟ — من الاستعلام إلى النمذجة

Chapter Thirty-One: Why Do We Need Mathematics? — From Querying to Modeling

الشكل 31 — الخريطة المفاهيمية للفصل

Figure 31 — Conceptual map of the chapter

في الموضع الحالي من الرحلة التعليمية، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

الشكل 31 — سير العمل الهندسي للفصل

Figure 31 — Engineering workflow of the chapter

This step does not start from scratch; it builds on what came before, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

في الأبواب السابقة، كانت علاقتك بالبيانات استعلامية (Query-based). تسأل قاعدة البيانات سؤالاً، فترجع لك صفوفاً. هذا النمط مثالي للإجابة عن أسئلة مثل "أعطني جميع طفرات BRCA1" أو "كم عدد العينات المريضة؟". لكن ماذا لو أردت أن تسأل سؤالاً مختلفاً تماماً؟

In previous units, your relationship with data was Query-based. You ask the database a question, and it returns rows. This pattern is ideal for answering questions like "Give me all BRCA1 mutations" or "How many diseased samples are there?". But what if you want to ask a completely different question?

مثال: "هل نمط التعبير الجيني في العينة (X) يشبه نمط التعبير في العينة (Y) أكثر من تشابهه مع العينة (Z)؟"

Example: "Is the gene expression pattern in sample (X) more similar to the pattern in sample (Y) than it is to sample (Z)؟"

SQL يمكنه إرجاع أعداد القراءات لكل جين، لكنه لا يستطيع إخبارك بالتشابه بين الأنماط. التشابه هو علاقة رياضية بين متجهين (Vectors). كل عينة تصبح نقطة في فضاء عالي الأبعاد، والتشابه يُقاس بالمسافة بين النقطتين.

SQL can return read counts for each gene, but it cannot tell you the similarity between patterns. Similarity is a mathematical relationship between two Vectors. Each sample becomes a point in a high-dimensional space, and similarity is measured by the distance between the points.

لماذا الرياضيات الآن؟

Why Mathematics Now?

لأن الأبواب القادمة (الحوسبة الكمومية) تعتمد كلياً على التمثيل الرياضي. لا يمكنك فهم Qubit دون فهم Vector، ولا يمكنك فهم Quantum Gate دون فهم Matrix. هذا الباب ليس

مجرد مراجعة للرياضيات المدرسية؛ هو إعادة تعريف للبيانات البيولوجية على أنها كائنات رياضية قابلة للتحويل والقياس.

Because the upcoming units (Quantum Computing) depend entirely on mathematical representation. You cannot understand Qubit without understanding Vector, and you cannot understand Quantum Gate without understanding Matrix. This unit is not just a review of school mathematics; it is a redefinition of biological data as mathematical objects that can be transformed and measured.

المسار:

The Path:

عينة واحدة (Sample) صف في جدول samples نقطة في فضاء (Vector)

جين واحد (Gene) صف في جدول genes بُعد (Dimension) في الفضاء

المفهوم (Concept) في SQL / OOP (In SQL / OOP) في الرياضيات (In Mathematics)

تعبير جيني (Expression) قيمة في عمود expression_count إحداثي (Coordinate) للمتجه

عينتان متشابهتان (Similar Samples) قيم متقاربة في عدة أعمدة

مسافة صغيرة بين متجهين (Euclidean/Cosine distance)

مجموعة عينات (Sample Set) جدول كامل مصفوفة (Matrix)

حيث كل صف عينة وكل عمود جين

هذا التحويل من "جدول" إلى "فضاء متجهي" هو جوهر علم البيانات (Data Science) و التعلم الآلي (Machine Learning)، وهو البوابة إلى النماذج التنبؤية والحوسبة المتقدمة.

This transformation from "Table" to "Vector Space" is the essence of Data Science and Machine Learning, and it is the gateway to predictive models and advanced computing.

الفصل الثاني والثلاثون: المتجهات (**Vectors**) — اللبنة الأساسية للبيانات متعددة الأبعاد

Chapter Thirty-Two: Vectors — The Basic Building Block of Multi-dimensional Data

الشكل 32 — الخريطة المفاهيمية للفصل

Figure 32 — Conceptual map of the chapter

بعد العمل السابق على الكائنات والبيانات، استخدمت الإحصاء والمصفوفات في تحليل البيانات؛ الآن ستبني فوق ذلك لغة رياضية للحسابات العلمية.

الشكل 32 — سير العمل الهندسي للفصل

Figure 32 — Engineering workflow of the chapter

At this point in the learning path, you used statistics and matrices to analyze data; now you will build on them as a mathematical language for scientific computation.

في الرياضيات، المتجه هو كائن رياضي له مقدار (Magnitude) واتجاه (Direction). في المعلوماتية الحيوية، المتجه هو تمثيل لمجموعة من القياسات لكيان واحد. كل عينة تصبح متجهاً.

In mathematics, a vector is a mathematical object with Magnitude and Direction. In bioinformatics, a vector is a representation of a set of measurements for a single entity. Each sample becomes a vector.

32.1 تمثيل عينة كمتجه (Sample as a Vector)

32.1 Representing a Sample as a Vector.

لنفترض أن لدينا 4 جينات: MYC ،EGFR ،TP53 ،BRCA1. العينة S001 لديها أعداد قراءات: [1200, 450, 2100, 800]. هذا هو متجه تعبير (Expression Vector) للعينة S001. يمكننا كتابته كـ:

v_S001 = (1200, 450, 2100, 800)

Suppose we have 4 genes: BRCA1, TP53, EGFR, MYC. Sample S001 has read counts: [1200, 450, 2100, 800]. This is the Expression Vector for sample S001. We can write it as:

v_S001 = (1200, 450, 2100, 800)

Representing this in OOP (Python):

تمثيل هذا في OOP (Python):

Python

```
from dataclasses import dataclass
from typing import List, Union
import math

@dataclass
class Vector:
    """تمثيل متجه رياضي مع عمليات أساسية.
    Mathematical Vector representation with basic operations.
    """
    components: List[float]

    def __post_init__(self):
        # تحويل جميع المكونات إلى float لضمان التوافق الحسابي
        # Convert all components to float to ensure computational compatibility
        self.components = [float(c) for c in self.components]
```

```
def __len__(self) -> int:
    """البعد (عدد المكونات).
    Dimension (number of components).
    """
    return len(self.components)

def __getitem__(self, index: int) -> float:
    """الوصول إلى مكون معين.
    Access a specific component.
    """
    return self.components[index]

def add(self, other: 'Vector') -> 'Vector':
    """جمع متجهين (عنصر بعنصر).
    Add two vectors (element-wise).
    """
    if len(self) != len(other):
        raise ValueError("لا يمكن جمع متجهات ذات أبعاد مختلفة")
    return Vector([a + b for a, b in zip(self.components, other.components)])

def scale(self, scalar: float) -> 'Vector':
    """ضرب المتجه في كمية قياسية.
    Multiply the vector by a scalar.
    """
    return Vector([c * scalar for c in self.components])

def magnitude(self) -> float:
    """حساب طول المتجه (المعيار الإقليدي).
    Calculate the vector magnitude (Euclidean norm).
    """
    return math.sqrt(sum(c**2 for c in self.components))
```

```
    def dot(self, other: 'Vector') -> float:
        """الضرب النقطي (Dot Product) بين متجهين.
        Dot product between two vectors.
        """
        if len(self) != len(other):
            raise ValueError("لا يمكن حساب الضرب النقطي لمتجهات ذات أبعاد مختلفة")
        return sum(a * b for a, b in zip(self.components, other.components))

    def cosine_similarity(self, other: 'Vector') -> float:
        """تشابه جيب التمام (Cosine Similarity) - مقياس تشابه بين -1 و 1.
        Cosine Similarity - similarity measure between -1 and 1.
        """
        if len(self) != len(other):
            raise ValueError("لا يمكن حساب التشابه لمتجهات ذات أبعاد مختلفة")
        mag_product = self.magnitude() * other.magnitude()
        if mag_product == 0:
            return 0.0
        return self.dot(other) / mag_product

    def euclidean_distance(self, other: 'Vector') -> float:
        """المسافة الإقليدية بين متجهين.
        Euclidean distance between two vectors.
        """
        if len(self) != len(other):
            raise ValueError("لا يمكن حساب المسافة لمتجهات ذات أبعاد مختلفة")
        return math.sqrt(sum((a - b)**2 for a, b in zip(self.components, other.components)))

    def __repr__(self) -> str:
        return f"Vector({self.components})"



# مثال: تمثيل عينتين كمتجهات
```

```
# Example: Representing two samples as vectors
sample1 = Vector([1200, 450, 2100, 800])  # S001
sample2 = Vector([980, 520, 1800, 750])   # S002
sample3 = Vector([100, 200, 80, 50])      # S003 (عينة مختلفة تماما)

# حساب التشابه بين العينات
print(f"التشابه (جيب التمام) بين S001 و S002: {sample1.cosine_similarity(sample2):.4f}")
print(f"التشابه بين S001 و S003: {sample1.cosine_similarity(sample3):.4f}")
print(f"المسافة الإقليدية بين S001 و S002: {sample1.euclidean_distance(sample2):.4f}")
```

ماذا يعني هذا بيولوجياً؟

- تشابه جيب التمام قريب من 1: العينتان لهما نفس نمط التعبير (نسب الجينات متشابهة)، حتى لو اختلف المقدار المطلق (عمق التسلسل).
- تشابه جيب التمام قريب من 0: العينتان مختلفتان تماماً في النمط.
- المسافة الإقليدية: تعكس الاختلاف في المقدار المطلق، متأثرة بعمق التسلسل.

What does this mean biologically?

· Cosine similarity close to 1: The two samples have the same expression pattern (gene ratios are similar), even if the absolute magnitude (sequencing depth) differs.
· Cosine similarity close to 0: The two samples are completely different in pattern.
· Euclidean distance: Reflects the difference in absolute magnitude, affected by sequencing depth.

32.2 المتجهات في سياق SQL —— استرجاع البيانات كمتجهات

32.2 Vectors in the SQL Context — Retrieving Data as Vectors.

الآن يمكننا ربط SQL بـ OOP لاسترجاع بيانات التعبير كمتجهات مباشرة.

Now we can link SQL to OOP to retrieve expression data as vectors directly.

Python

```
class ExpressionVectorRepository:
    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path

    def get_sample_vector(self, sample_id: int, gene_list: List[str]) -> Optional[Vector]:
        """استرجاع متجه تعبير لعينة معينة بناءً على قائمة جينات.
        Retrieve expression vector for a specific sample based on a gene list.
        """
        if not gene_list:
            return None

        # بناء استعلام ديناميكي لجلب القراءات لجينات محددة
        placeholders = ','.join(['?'] * len(gene_list))
        query = f"""
            SELECT
                seq.gene_symbol,
                rc.expression_count
            FROM read_counts rc
            JOIN sequences seq ON rc.gene_symbol = seq.gene_symbol
            WHERE rc.sample_id = ? AND seq.gene_symbol IN ({placeholders})
            ORDER BY seq.gene_symbol
        """

        with sqlite3.connect(self.db_path) as conn:
            conn.row_factory = sqlite3.Row
            cursor = conn.cursor()
            params = [sample_id] + gene_list
```

```
        rows = cursor.execute(query, params).fetchall()

        if len(rows) != len(gene_list):
            raise ValueError(f"بعض الجينات غير موجودة للعينة {sample_id}")

        # ترتيب القيم حسب ترتيب gene_list الأصلي
        value_map = {row["gene_symbol"]: row["expression_count"] for row in rows}
        components = [float(value_map.get(gene, 0.0)) for gene in gene_list]

        return Vector(components)

    def get_all_sample_vectors(self, gene_list: List[str]) -> Dict[int, Vector]:
        """استرجاع متجهات التعبير لجميع العينات.
        Retrieve expression vectors for all samples.
        """
        with sqlite3.connect(self.db_path) as conn:
            # جلب جميع معرفات العينات
            cursor = conn.cursor()
            sample_ids = [row[0] for row in cursor.execute("SELECT sample_id FROM
samples").fetchall()]

        # بناء متجه لكل عينة
        vectors = {}
        for sid in sample_ids:
            vec = self.get_sample_vector(sid, gene_list)
            if vec:
                vectors[sid] = vec

        return vectors
```

الفصل الثالث والثلاثون: المصفوفات (**Matrices**) — مجموعة من المتجهات

Chapter Thirty-Three: Matrices — A Collection of Vectors

الشكل 33 — الخريطة المفاهيمية للفصل

Figure 33 — Conceptual map of the chapter

الأساس الذي تم بناؤه في الجزء الثاني يسمح لنا الآن بـ استخدمت الإحصاء والمصفوفات في تحليل البيانات؛ الآن ستبني فوق ذلك لغة رياضية للحسابات العلمية.

الشكل 33 — سير العمل الهندسي للفصل

Figure 33 — Engineering workflow of the chapter

After working with objects and data in the previous part, you used statistics and matrices to analyze data; now you will build on them as a mathematical language for scientific computation.

إذا كان المتجه يمثل عينة واحدة، فالمصفوفة (Matrix) تمثل مجموعة من العينات (صفوف) وجينات (أعمدة). المصفوفة

هي البنية الأساسية لمعظم تحليلات المعلوماتية الحيوية: كل صف عينة، كل عمود جين، وكل خلية هي قيمة التعبير.

If a vector represents one sample, a Matrix represents a collection of samples (rows) and genes (columns). The matrix is the fundamental structure for most bioinformatics analyses: each row is a sample, each column is a gene, and each cell is the expression value.

33.1 تمثيل المصفوفة في OOP

33.1 Representing a Matrix in OOP.

Python

```
@dataclass
class Matrix:
    """تمثيل مصفوفة رياضية مع عمليات أساسية.
    Mathematical Matrix representation with basic operations.
    """
    rows: List[List[float]]  # قائمة من القوائم (كل قائمة تمثل صفاً)

    def __post_init__(self):
        #  تحويل جميع القيم إلى float
```

```
        self.rows = [[float(cell) for cell in row] for row in self.rows]

    def shape(self) -> tuple:
        """إرجاع أبعاد المصفوفة (عدد الصفوف, عدد الأعمدة).
        Return matrix shape (number of rows, number of columns).
        """
        if not self.rows:
            return (0, 0)
        return (len(self.rows), len(self.rows[0]))

    def get_row(self, index: int) -> Vector:
        """استرجاع صف معين كمتجه.
        Retrieve a specific row as a Vector.
        """
        if index >= len(self.rows):
            raise IndexError("Index out of range")
        return Vector(self.rows[index])

    def get_column(self, index: int) -> Vector:
        """استرجاع عمود معين كمتجه.
        Retrieve a specific column as a Vector.
        """
        if not self.rows or index >= len(self.rows[0]):
            raise IndexError("Index out of range")
        return Vector([row[index] for row in self.rows])

    def transpose(self) -> 'Matrix':
        """تبديل المصفوفة (الأعمدة تصبح صفوفاً والعكس).
        Transpose the matrix (columns become rows and vice versa).
        """
        if not self.rows:
            return Matrix([])
```

```
        n_rows, n_cols = self.shape()
        transposed = [[self.rows[r][c] for r in range(n_rows)] for c in range(n_cols)]
        return Matrix(transposed)

    def add(self, other: 'Matrix') -> 'Matrix':
        """جمع مصفوفتين (عنصر بعنصر).
        Add two matrices (element-wise).
        """
        if self.shape() != other.shape():
            raise ValueError("لا يمكن جمع مصفوفات ذات أبعاد مختلفة")
        n_rows, n_cols = self.shape()
        result = [
            [self.rows[i][j] + other.rows[i][j] for j in range(n_cols)]
            for i in range(n_rows)
        ]
        return Matrix(result)

    def multiply(self, other: 'Matrix') -> 'Matrix':
        """ضرب مصفوفتين (الضرب المصفوفي القياسي).
        Multiply two matrices (standard matrix multiplication).
        """
        n_rows_self, n_cols_self = self.shape()
        n_rows_other, n_cols_other = other.shape()

        if n_cols_self != n_rows_other:
            raise ValueError(
                f"لا يمكن ضرب مصفوفات: أعمدة الأولى ({n_cols_self}) != صفوف الثانية ({n_rows_other})"
            )

        result = []
        for i in range(n_rows_self):
            row = []
```

```
        for j in range(n_cols_other):
            # الضرب النقطي للصف i مع العمود j في الأولى في الثانية
            dot_product = sum(
                self.rows[i][k] * other.rows[k][j]
                for k in range(n_cols_self)
            )
            row.append(dot_product)
        result.append(row)

    return Matrix(result)

def scale(self, scalar: float) -> 'Matrix':
    """ضرب جميع عناصر المصفوفة بكمية قياسية.
    Multiply all matrix elements by a scalar.
    """
    n_rows, n_cols = self.shape()
    result = [
        [self.rows[i][j] * scalar for j in range(n_cols)]
        for i in range(n_rows)
    ]
    return Matrix(result)

def to_dataframe(self):
    """تحويل إلى DataFrame لسهولة التصور (يتطلب pandas).
    Convert to DataFrame for easy visualization (requires pandas).
    """
    try:
        import pandas as pd
        return pd.DataFrame(self.rows)
    except ImportError:
        return self.rows
```

```
    def __repr__(self) -> str:
        return f"Matrix({self.rows})"
```

33.2 المصفوفة الجينومية: جدول التعبير كـ Matrix

33.2 The Genomic Matrix: Expression Table as a Matrix.

في مشروع RNA-seq، لدينا مصفوفة تعبير حجمها (عينات × جينات). هذه المصفوفة هي أساس تحليل PCA، والتجميع (Clustering)، والعديد من خوارزميات التعلم الآلي.

In an RNA-seq project, we have an expression matrix of size (samples × genes). This matrix is the foundation for PCA analysis, Clustering, and many machine learning algorithms.

Python

```
class ExpressionMatrixBuilder:
    """بناء مصفوفة التعبير من قاعدة البيانات.
    Build expression matrix from the database.
    """
```

```
    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path

    def build_matrix(self, sample_ids: List[int], gene_list: List[str]) -> Matrix:
        """بناء مصفوفة التعبير لعينات وجينات محددة.
        Build expression matrix for specific samples and genes.
        """
        if not sample_ids or not gene_list:
            return Matrix([])

        matrix_rows = []
        for sid in sample_ids:
            # استخدام VectorRepository السابق
            vec_repo = ExpressionVectorRepository(self.db_path)
            vec = vec_repo.get_sample_vector(sid, gene_list)
            if vec:
                matrix_rows.append(vec.components)

        return Matrix(matrix_rows)

    def build_complete_matrix(self) -> Matrix:
        """بناء مصفوفة كاملة لكل العينات وكل الجينات (قد يكون ثقيلاً).
        Build complete matrix for all samples and all genes (may be heavy).
        """
        with sqlite3.connect(self.db_path) as conn:
            cursor = conn.cursor()
            # جلب جميع العينات
            sample_ids = [row[0] for row in cursor.execute("SELECT sample_id FROM
samples").fetchall()]
            # جلب جميع الجينات (من جدول read_counts)
            gene_list = [row[0] for row in cursor.execute(
                "SELECT DISTINCT gene_symbol FROM read_counts ORDER BY gene_symbol"
```

```
        ).fetchall()]

        return self.build_matrix(sample_ids, gene_list)

# مثال: بناء المصفوفة وتحليلها
# Example: Build matrix and analyze it
# builder = ExpressionMatrixBuilder()
# matrix = builder.build_complete_matrix()
# print(f"مصفوفة التعبير: {matrix.shape()[0]} عينة × {matrix.shape()[1]} جين")
```

الفصل الرابع والثلاثون: الدوال (**Functions**) — تحويل البيانات من صورة إلى أخرى

Chapter Thirty-Four: Functions — Transforming Data from One Form to Another

الشكل 34 — الخريطة المفاهيمية للفصل

Figure 34 — Conceptual map of the chapter

هنا يتحول ما تعلمته سابقًا إلى طبقة جديدة من التصميم، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

الشكل 34 — سير العمل الهندسي للفصل

Figure 34 — Engineering workflow of the chapter

The foundation built in Part Two now allows you to you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

في البرمجة، الدالة تأخذ مدخلات وتنتج مخرجات. في الرياضيات، الدالة تفعل الشيء نفسه ولكن بشكل تجريدي. في المعلوماتية الحيوية، الدوال الرياضية هي ما يسمح لنا بتحويل بياناتنا: التطبيع (Normalization)، تحويل اللوغاريتم، والتصفية.

In programming, a function takes inputs and produces outputs. In mathematics, a function does the same but abstractly. In bioinformatics, mathematical functions are what allow us to transform our data: Normalization, Logarithmic Transformation, and Filtering.

34.1 التطبيع كدالة رياضية

34.1 Normalization as a Mathematical Function.

دالة التطبيع الأكثر شيوعاً هي تطبيع Z-score (Standardization):

تحويل البيانات بحيث يكون المتوسط = 0 والانحراف المعياري = 1.

The most common normalization function is Z-score Normalization (Standardization): transforming data so that the mean = 0 and standard deviation = 1.

$$Z(x) = (x - \mu) / \sigma$$

حيث μ هو المتوسط و σ هو الانحراف المعياري.

Where μ is the mean and σ is the standard deviation.

Python

```
from typing import Callable


class MatrixTransformer:
    """تطبيق دوال رياضية على المصفوفات.
    Apply mathematical functions to matrices.
    """
```

```
@staticmethod
def z_score_normalize(matrix: Matrix) -> Matrix:
    """تطبيق تطبيع Z-score على حدة (جين) على كل عمود.
    Apply Z-score normalization to each column (gene) independently.
    """
    n_rows, n_cols = matrix.shape()
    if n_rows == 0:
        return matrix

    normalized = []
    for col_idx in range(n_cols):
        column = matrix.get_column(col_idx).components
        mean = sum(column) / len(column)
        variance = sum((x - mean) ** 2 for x in column) / len(column)
        std = math.sqrt(variance) if variance > 0 else 1.0

        z_scored = [(x - mean) / std for x in column]
        normalized.append(z_scored)

     # إعادة بناء المصفوفة (الأعمدة المصححة أصبحت صفوفاً في القائمة، لذا نحتاج إلى تبديل)
     # نعيد ترتيبها إلى صفوف (عينات) وأعمدة (جينات)
    transposed = Matrix(normalized).transpose()
    return transposed

@staticmethod
def log2_transform(matrix: Matrix, pseudo_count: float = 1.0) -> Matrix:
    """تطبيق تحويل لوغاريتمي (قاعدة 2) مع إضافة قيمة صغيرة لتجنب الصفر.
    Apply logarithmic transformation (base 2) with a small pseudo-count to avoid zero.
    """
    n_rows, n_cols = matrix.shape()
    if n_rows == 0:
```

```
        return matrix

    transformed = [
        [math.log2(cell + pseudo_count) for cell in row]
        for row in matrix.rows
    ]
    return Matrix(transformed)

@staticmethod
def apply_function(matrix: Matrix, func: Callable[[float], float]) -> Matrix:
    """تطبيق دالة رياضية عامة على كل عنصر.
    Apply a generic mathematical function to every element.
    """
    n_rows, n_cols = matrix.shape()
    if n_rows == 0:
        return matrix

    transformed = [
        [func(cell) for cell in row]
        for row in matrix.rows
    ]
    return Matrix(transformed)


# مثال: تطبيق دوال مختلفة على مصفوفة التعبير
# Example: Applying different functions to the expression matrix
# matrix = ExpressionMatrixBuilder().build_complete_matrix()
# z_scored = MatrixTransformer.z_score_normalize(matrix)
# log_transformed = MatrixTransformer.log2_transform(matrix)
```

الفصل الخامس والثلاثون: الاحتمال (**Probability**) — قياس عدم اليقين في البيانات البيولوجية

Chapter Thirty-Five: Probability — Measuring Uncertainty in Biological Data

الشكل 35 — الخريطة المفاهيمية للفصل

Figure 35 — Conceptual map of the chapter

الشكل 35 — سير العمل الهندسي للفصل

Figure 35 — Engineering workflow of the chapter

في الجزء الثاني (الفصل 49)، تعلمت عن التوزيعات الاحتمالية (Normal، Poisson، Negative Binomial). هذا الفصل يعيد النظر فيها من منظور النمذجة الرياضية. الاحتمال هو لغتنا للتعامل مع عدم اليقين: أخطاء التسلسل، التباين البيولوجي، وعشوائية أخذ العينات.

In Part Two (Chapter 49), you learned about probability distributions (Normal, Poisson, Negative Binomial). This chapter revisits them from the perspective of Mathematical Modeling. Probability is our language for dealing with uncertainty: sequencing errors, biological variation, and sampling randomness.

35.1 تمثيل التوزيعات الاحتمالية في OOP

35.1 Representing Probability Distributions in OOP.

Python

```
from dataclasses import dataclass
import math
from typing import List, Optional


@dataclass
class Distribution:
    """فئة أساسية للتوزيعات الاحتمالية.
    Base class for probability distributions.
    """
    name: str


    def pdf(self, x: float) -> float:
        """دالة الكثافة الاحتمالية (Probability Density Function).
```

```
        Probability Density Function.
        """
        raise NotImplementedError

    def cdf(self, x: float) -> float:
        """دالة التوزيع التراكمي (Cumulative Distribution Function).
        Cumulative Distribution Function.
        """
        raise NotImplementedError

    def sample(self, n: int = 1) -> List[float]:
        """توليد عينات عشوائية من التوزيع.
        Generate random samples from the distribution.
        """
        raise NotImplementedError

@dataclass
class NormalDistribution(Distribution):
    """التوزيع الطبيعي (Gaussian).
    Normal (Gaussian) Distribution.
    """
    mean: float
    std: float

    def __post_init__(self):
        self.name = f"Normal(μ={self.mean}, σ={self.std})"

    def pdf(self, x: float) -> float:
        """حساب دالة الكثافة عند النقطة x.
        Calculate PDF at point x.
        """
        return (1 / (self.std * math.sqrt(2 * math.pi))) * math.exp(
```

```
            -((x - self.mean) ** 2) / (2 * self.std ** 2)
        )

    def sample(self, n: int = 1) -> List[float]:
        """توليد عينات باستخدام Box-Muller (محاكاة بسيطة).
        Generate samples using Box-Muller (simple simulation).
        """
        # هذه محاكاة بسيطة، في الواقع نستخدم random.gauss أو numpy
        import random
        return [random.gauss(self.mean, self.std) for _ in range(n)]


@dataclass
class PoissonDistribution(Distribution):
    """توزيع بواسون (للأعداد الصحيحة النادرة).
    Poisson Distribution (for rare integer counts).
    """
    lambda_: float  # المعدل (mean = variance = lambda)

    def __post_init__(self):
        self.name = f"Poisson(λ={self.lambda_})"

    def pdf(self, k: int) -> float:
        """دالة الكتلة الاحتمالية (لأنها متغير منفصل).
        Probability Mass Function (discrete variable).
        """
        if k < 0:
            return 0.0
        return (math.exp(-self.lambda_) * (self.lambda_ ** k)) / math.factorial(k)

    def sample(self, n: int = 1) -> List[int]:
        """توليد عينات باستخدام محاكاة بواسون (بسيطة).
        Generate samples using simple Poisson simulation.
```

```
        """
        import random
        return [random.poisson(self.lambda_) for _ in range(n)]

@dataclass
class NegativeBinomialDistribution(Distribution):
    """التوزيع السالب ذو الحدين (لتشتت بيانات RNA-seq).
    Negative Binomial Distribution (for RNA-seq overdispersion).
    """
    mean: float
    dispersion: float  # معلمة التشتت (كلما زادت، زاد التباين)

    def __post_init__(self):
        self.name = f"NegBinom(μ={self.mean}, φ={self.dispersion})"

    def variance(self) -> float:
        """حساب التباين (mean + dispersion * mean^2).
        Calculate variance (mean + dispersion * mean^2).
        """
        return self.mean + self.dispersion * (self.mean ** 2)

    def pdf(self, k: int) -> float:
        """محاكاة بسيطة (في الواقع نستخدم scipy.stats.nbinom).
        Simple simulation (in practice, use scipy.stats.nbinom).
        """
        # هذه محاكاة تقريبية للتبسيط التعليمي
        # This is an approximation for educational simplicity
        import math
        if k < 0:
            return 0.0
        # معامل التشتت يتحكم في الشكل
        r = 1 / self.dispersion if self.dispersion > 0 else 10
```

```
        p = r / (r + self.mean) if r + self.mean > 0 else 0.5
        return (math.comb(k + r - 1, k) * (p ** r) * ((1 - p) ** k))

    def sample(self, n: int = 1) -> List[int]:
        """محاكاة (في الواقع نستخدم numpy.random.negative_binomial).
        Simulation (in practice, use numpy.random.negative_binomial).
        """
        import random
        r = 1 / self.dispersion if self.dispersion > 0 else 10
        p = r / (r + self.mean) if r + self.mean > 0 else 0.5
        return [random.negative_binomial(r, p) for _ in range(n)]
```

35.2 ربط الاحتمال بقاعدة البيانات — توزيع التعبير الجيني

35.2 Linking Probability to the Database — Gene Expression Distribution.

Python

```
class DistributionFitter:
    """ملاءمة التوزيعات الاحتمالية لبيانات التعبير.
    Fitting probability distributions to expression data.
    """
    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path

    def get_gene_expression_values(self, gene_symbol: str) -> List[float]:
        """استرجاع جميع قيم التعبير لجين معين عبر جميع العينات.
        Retrieve all expression values for a specific gene across all samples.
```

```
    """
    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.cursor()
        cursor.execute(
            "SELECT expression_count FROM read_counts WHERE gene_symbol = ?",
            (gene_symbol,)
        )
        return [float(row[0]) for row in cursor.fetchall()]

def fit_poisson(self, gene_symbol: str) -> Optional[PoissonDistribution]:
    """ملاءمة توزيع بواسون للجين (تقدير lambda من المتوسط).
    Fit Poisson distribution to the gene (estimate lambda from mean).
    """
    values = self.get_gene_expression_values(gene_symbol)
    if not values:
        return None
    mean = sum(values) / len(values)
    return PoissonDistribution(lambda_=mean)

def fit_negative_binomial(self, gene_symbol: str) -> Optional[NegativeBinomialDistribution]:
    """ملاءمة توزيع سالب ذو حدين (تقدير التشتت من التباين).
    Fit Negative Binomial (estimate dispersion from variance).
    """
    values = self.get_gene_expression_values(gene_symbol)
    if not values or len(values) < 2:
        return None

    mean = sum(values) / len(values)
    variance = sum((v - mean) ** 2 for v in values) / len(values)

    if mean == 0:
        return NegativeBinomialDistribution(mean=mean, dispersion=0.0)
```

```
        # تقدير تقريبي للتشتت: (variance - mean) / mean^2
        dispersion = max(0.0, (variance - mean) / (mean ** 2))
        return NegativeBinomialDistribution(mean=mean, dispersion=dispersion)


# مثال: تحليل توزيع جين معين
# Example: Analyzing distribution of a specific gene
# fitter = DistributionFitter()
# nb = fitter.fit_negative_binomial("BRCA1")
# print(f"توزيع BRCA1: متوسط={nb.mean:.2f}, تشتت={nb.dispersion:.4f}")
```

الفصل السادس والثلاثون: الإحصاء (**Statistics**) — من الوصف إلى الاستدلال

Chapter Thirty-Six: Statistics — From Description to Inference

الشكل 36 — الخريطة المفاهيمية للفصل

Figure 36 — Conceptual map of the chapter

الشكل 36 — سير العمل الهندسي للفصل

Figure 36 — Engineering workflow of the chapter

الخبرة السابقة مع الملفات والكائنات تصبح في هذا الفصل مدخلًا إلى، تعلمت الإحصاء الوصفي (المتوسط، الوسيط، الانحراف المعياري). في هذا الفصل، ننتقل إلى الإحصاء الاستدلالي (Inferential Statistics): استخدام العينات لاستنتاج خصائص المجتمع الأكبر، واختبار الفرضيات، وتقدير فترات الثقة. هذا هو الجسر بين "ما نراه في بياناتنا" و "ما يمكننا استنتاجه عن العالم الحقيقي".

Here, an earlier skill becomes a new layer of system design, you learned descriptive statistics (mean, median, standard deviation). In this chapter, we move to Inferential Statistics: using samples to infer properties of the larger population, testing hypotheses, and estimating confidence intervals. This is

the bridge between "what we see in our data" and "what we can infer about the real world".

36.1 اختبار t كدالة رياضية (مع OOP)

36.1 T-Test as a Mathematical Function (with OOP).

الآن نستخدم المهارة السابقة للوصول إلى مستوى جديد من النظام. استخدمنا اختبار t لمقارنة مجموعتين. الآن نعيد تنفيذه كدالة رياضية قابلة لإعادة الاستخدام في نظامنا.

Your previous experience with files and objects now becomes an entry point to we used the t-test to compare two groups. Now we re-implement it as a reusable mathematical function in our system.

Python

```
@dataclass
class TTestResult:
    """نتيجة اختبار t.
    T-Test result.
    """
    t_statistic: float
    p_value: float
    df: int  # درجات الحرية (Degrees of freedom)
```

```
    mean_diff: float
    ci_lower: float
    ci_upper: float

def independent_t_test(sample1: Vector, sample2: Vector, alpha: float = 0.05) ->
Optional[TTestResult]:
    """
    اختبار t لعينتين مستقلتين (Independent Two-Sample T-Test).
    Independent two-sample T-Test (Welch's t-test).
    """
    if len(sample1) < 2 or len(sample2) < 2:
        return None

    n1, n2 = len(sample1), len(sample2)
    mean1 = sum(sample1.components) / n1
    mean2 = sum(sample2.components) / n2
    mean_diff = mean1 - mean2

    # حساب التباين لكل عينة
    var1 = sum((x - mean1) ** 2 for x in sample1.components) / (n1 - 1)
    var2 = sum((x - mean2) ** 2 for x in sample2.components) / (n2 - 1)

    # اختبار t لـ Welch (لا يفترض تساوي التباينات)
    se = math.sqrt((var1 / n1) + (var2 / n2))
    if se == 0:
        return None

    t_stat = mean_diff / se

    # درجات الحرية (Satterthwaite approximation)
    df_numerator = ((var1 / n1) + (var2 / n2)) ** 2
    df_denominator = ((var1 / n1) ** 2 / (n1 - 1)) + ((var2 / n2) ** 2 / (n2 - 1))
```

```
df = df_numerator / df_denominator if df_denominator > 0 else n1 + n2 - 2

 # حساب قيمة p (توزيع t)
 # هذه محاكاة، في الواقع نستخدم scipy.stats.t
try:
    from scipy import stats
    p_value = stats.t.sf(abs(t_stat), df) * 2  # اختبار ثنائي الطرف
    ci_lower = mean_diff - stats.t.ppf(1 - alpha/2, df) * se
    ci_upper = mean_diff + stats.t.ppf(1 - alpha/2, df) * se
except ImportError:
     # تقريب بسيط إذا لم يكن scipy متاحاً (للتعليم فقط)
    p_value = 0.5  # قيمة وهمية
    ci_lower = mean_diff - 1.96 * se
    ci_upper = mean_diff + 1.96 * se

return TTestResult(
    t_statistic=t_stat,
    p_value=p_value,
    df=df,
    mean_diff=mean_diff,
    ci_lower=ci_lower,
    ci_upper=ci_upper
)
```

36.2 ربط الإحصاء بقاعدة البيانات — مقارنة مجموعات العينات

36.2 Linking Statistics to the Database — Comparing Sample Groups.

Python

```
class StatisticalAnalysisService:
    """خدمة التحليل الإحصائي باستخدام البيانات من قاعدة البيانات.
    Statistical Analysis Service using data from the database.
    """
    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path
        self.vec_repo = ExpressionVectorRepository(db_path)

    def compare_groups(self, gene_symbol: str, group1_condition: str, group2_condition: str) ->
Optional[TTestResult]:
        """مقارنة تعبير جين معين بين مجموعتين من العينات (حسب الحالة).
        Compare expression of a specific gene between two groups of samples (by condition).
        """
        # جلب معرفات العينات لكل مجموعة
        with sqlite3.connect(self.db_path) as conn:
            cursor = conn.cursor()
            cursor.execute(
                "SELECT sample_id FROM samples WHERE status = ?",
                (group1_condition,)
            )
            group1_ids = [row[0] for row in cursor.fetchall()]

            cursor.execute(
                "SELECT sample_id FROM samples WHERE status = ?",
                (group2_condition,)
            )
            group2_ids = [row[0] for row in cursor.fetchall()]
```

```
        if not group1_ids or not group2_ids:
            return None

        # جلب قيم التعبير للجين المطلوب لكل مجموعة
        # ملاحظة: هذا يفترض أن لدينا جدول read_counts مع أعمدة (sample_id, gene_symbol,
expression_count)
        def get_expression_for_samples(sample_ids: List[int], gene: str) -> List[float]:
            with sqlite3.connect(self.db_path) as conn:
                placeholders = ','.join(['?'] * len(sample_ids))
                query = f"""
                    SELECT expression_count
                    FROM read_counts
                    WHERE sample_id IN ({placeholders}) AND gene_symbol = ?
                """
                cursor = conn.cursor()
                params = sample_ids + [gene]
                return [float(row[0]) for row in cursor.execute(query, params).fetchall()]

        group1_vals = get_expression_for_samples(group1_ids, gene_symbol)
        group2_vals = get_expression_for_samples(group2_ids, gene_symbol)

        if len(group1_vals) < 2 or len(group2_vals) < 2:
            return None

        # تحويل إلى متجهات
        v1 = Vector(group1_vals)
        v2 = Vector(group2_vals)

        return independent_t_test(v1, v2)

# مثال: مقارنة تعبير BRCA1 بين المرضى والأصحاء
```

```
# Example: Comparing BRCA1 expression between diseased and healthy
# service = StatisticalAnalysisService()
# result = service.compare_groups("BRCA1", "diseased", "control")
# if result:
#   print(f"t = {result.t_statistic:.4f}, p = {result.p_value:.4f}")
```

الفصل السابع والثلاثون: الجبر الخطي (**Linear Algebra**) — لغة الأنظمة

Chapter Thirty-Seven: Linear Algebra — The Language of Systems

الشكل 37 — الخريطة المفاهيمية للفصل

Figure 37 — Conceptual map of the chapter

في هذه المرحلة من المسار، استخدمت الإحصاء والمصفوفات في تحليل البيانات؛ الآن ستبني فوق ذلك لغة رياضية للحسابات العلمية.

الشكل 37 — سير العمل الهندسي للفصل

Figure 37 — Engineering workflow of the chapter

We now use the earlier skill to reach the next level of the system. you used statistics and matrices to analyze data; now you will build on them as a mathematical language for scientific computation.

الجبر الخطي هو الرياضيات التي تصف التحولات و العلاقات في الفضاءات المتجهية. في المعلوماتية الحيوية، الجبر

الخطي هو ما يسمح لنا بتقليل الأبعاد (PCA)، وتجميع العينات (Clustering)، وفهم العلاقات بين الجينات. هذا هو الباب الذي يفتح الطريق إلى الحوسبة الكمومية، حيث كل شيء هو تحويل خطي على متجهات في فضاء هيلبرت.

Linear Algebra is the mathematics that describes Transformations and Relationships in vector spaces. In bioinformatics, linear algebra is what allows us to reduce dimensions (PCA), cluster samples (Clustering), and understand relationships between genes. This is the unit that opens the door to Quantum Computing, where everything is a linear transformation on vectors in Hilbert space.

37.1 التحويلات الخطية (Linear Transformations)

37.1 Linear Transformations.

التحويل الخطي هو دالة تحافظ على جمع المتجهات والضرب في كمية قياسية: T(a*v + b*w) = a*T(v) + b*T(w). في عالم المصفوفات، كل تحويل خطي يُنمذج بضرب مصفوفة في متجه.

A linear transformation is a function that preserves vector addition and scalar multiplication: T(a*v + b*w) = a*T(v) + b*T(w). In the world of matrices, every linear transformation is modeled by multiplying a matrix by a vector.

مثال: تحويل PCA (تقليل الأبعاد)

Example: PCA Transformation (Dimensionality Reduction).

تحويل PCA هو تحويل خطي يأخذ مصفوفة التعبير (عينات × جينات) ويضربها في مصفوفة تحويل (جينات × مكونات) لإنتاج مصفوفة جديدة (عينات × مكونات) ذات أبعاد أقل.

PCA transformation is a linear transformation that takes the expression matrix (samples × genes) and multiplies it by a transformation matrix (genes × components) to produce a new matrix (samples × components) with fewer dimensions.

Python

```
class LinearAlgebraService:
    """خدمات الجبر الخطي للتطبيقات البيولوجية.
    Linear Algebra services for biological applications.
    """

    @staticmethod
    def pca_transform(matrix: Matrix, n_components: int = 2) -> Matrix:
        """
        تحويل PCA مبسط (محاكاة تعليمية).
        Simplified PCA transformation (educational simulation).

        في الواقع، يتم حساب المتجهات الذاتية (Eigenvectors) لمصفوفة التغاير.
        In reality, eigenvectors of the covariance matrix are computed.
        """
        n_rows, n_cols = matrix.shape()
        if n_components > n_cols:
            n_components = n_cols

        # 1. مركزية البيانات (طرح المتوسط من كل عمود)
        # 1. Center data (subtract mean from each column)
        centered = []
        for col_idx in range(n_cols):
            col = matrix.get_column(col_idx).components
```

```
        mean = sum(col) / len(col)
        centered_col = [x - mean for x in col]
        centered.append(centered_col)

    # إعادة بناء المصفوفة المركزية (صفوف = عينات، أعمدة = جينات)
    centered_matrix = Matrix(centered).transpose()

    # 2. محاكاة: نأخذ أول n_components أعمدة كـ "مكونات رئيسية" (لتبسيط التعليم)
    # 2. Simulation: take the first n_components columns as "principal components" (for educational simplicity)
    # في الواقع، هذه هي المكونات الرئيسية الفعلية بعد تحليل القيم الذاتية
    # In reality, these are the actual principal components after eigenvalue decomposition
    pca_result = []
    for row in centered_matrix.rows:
        # تطبيق تحويل وهمي: كل مكون هو متوسط مرجح للأبعاد
        transformed_row = []
        for comp in range(n_components):
            # وزن عشوائي تعليمي (في الواقع، هو المتجه الذاتي)
            weight = [1.0 / (comp + 1) for _ in range(n_cols)]
            value = sum(row[i] * weight[i] for i in range(n_cols)) / n_cols
            transformed_row.append(value)
        pca_result.append(transformed_row)

    return Matrix(pca_result)
```

37.2 القيم الذاتية (Eigenvalues) والمتجهات الذاتية (Eigenvectors)

37.2 Eigenvalues and Eigenvectors.

القيم الذاتية والمتجهات الذاتية هي جوهر تحليل المصفوفات. المتجه الذاتي هو متجه لا يغير اتجاهه عند تطبيق التحويل الخطي، بل يتغير مقداره فقط (بقيمة eigenvalue). في المعلوماتية الحيوية، المتجهات الذاتية لمصفوفة التغاير هي "المكونات الرئيسية" في PCA، والتي تكشف عن أنماط التباين الخفية في البيانات.

Eigenvalues and Eigenvectors are the heart of matrix analysis. An eigenvector is a vector that does not change direction when the linear transformation is applied, only its magnitude changes (by the eigenvalue). In bioinformatics, the eigenvectors of the covariance matrix are the "principal components" in PCA, which reveal hidden patterns of variation in the data.

مفهوم المتجه الذاتي في سياق التعبير الجيني:

Eigenvector Concept in Gene Expression Context:

إذا كان لدينا مصفوفة تعبير، فإن المتجه الذاتي الأول يمثل "نمط التعبير" الذي يفسر أكبر قدر من التباين بين العينات. هذا النمط قد يكون مرتبطاً بالحالة المرضية، أو بنوع النسيج، أو بعامل تقني (مثل دفعة التسلسل).

If we have an expression matrix, the first eigenvector represents the "expression pattern" that explains the largest amount of variance between samples. This pattern might be associated with disease state, tissue type, or a technical factor (such as sequencing batch).

Python

```
class EigenDecomposition:
    """
    تحليل القيم الذاتية والمتجهات الذاتية (محاكاة تعليمية).
    Eigenvalue and eigenvector decomposition (educational simulation).
    في الواقع، نستخدم numpy.linalg.eig أو scipy.linalg.eig.
    In reality, we use numpy.linalg.eig or scipy.linalg.eig.
    """
    @staticmethod
    def simulate_eigen_analysis(matrix: Matrix, n_components: int = 2) -> dict:
```

```
"""
محاكاة تعليمية لتحليل القيم الذاتية.
Educational simulation of eigenvalue analysis.
"""
n_rows, n_cols = matrix.shape()
if n_rows == 0 or n_cols == 0:
    return {"eigenvalues": [], "eigenvectors": []}

# محاكاة: نحسب التباين في البيانات (نتجاهل التغاير للتبسيط)
# Simulation: calculate variance in data (ignore covariance for simplicity)
variances = []
for col_idx in range(n_cols):
    col = matrix.get_column(col_idx).components
    mean = sum(col) / len(col)
    var = sum((x - mean) ** 2 for x in col) / len(col)
    variances.append(var)

# ترتيب التباينات تنازلياً (محاكاة للقيم الذاتية)
sorted_variances = sorted(variances, reverse=True)
eigenvalues = sorted_variances[:n_components]

# محاكاة المتجهات الذاتية (مصفوفة هوية مرجحة)
# Simulation of eigenvectors (weighted identity matrix)
eigenvectors = []
for i in range(n_components):
    # متجه طوله n_cols مع 1 في الموضع i و 0 في الباقي (مبسط)
    vec = [0.0] * n_cols
    if i < n_cols:
        vec[i] = 1.0
    eigenvectors.append(vec)

return {
```

```
        "eigenvalues": eigenvalues,
        "eigenvectors": eigenvectors,
        "explained_variance_ratio": [
            ev / sum(variances) for ev in eigenvalues
        ]
    }
```

الفصل السابع والثلاثون والنصف (37.5): من الإحصاء القياسي إلى الفضاءات المتجهية — الجسر المفاهيمي

Chapter 37.5: From Scalar Statistics to Vector Spaces — The Conceptual Bridge

الشكل 38 — الخريطة المفاهيمية للفصل

Figure 38 — Conceptual map of the chapter

الشكل 38 — سير العمل الهندسي للفصل

Figure 38 — Engineering workflow of the chapter

لماذا ننتقل من أرقام فردية (متوسط، انحراف معياري) إلى متجهات ومصفوفات؟

Why do we move from single numbers (mean, standard deviation) to vectors and matrices؟

هذا هو السؤال الجوهري الذي يربط الجزء الثاني (الإحصاء) بالجزء الثالث (الجبر الخطي والكم). الإجابة تكمن في طبيعة البيانات الجينومية نفسها.

This is the fundamental question that links Part Two (Statistics) to Part Three (Linear Algebra and Quantum). The answer lies in the nature of genomic data itself.

عند الانتقال من الجزء السابق إلى هذا الفصل، كنت تنظر إلى جين واحد في كل مرة: متوسط تعبير جين BRCA1، تباين تعبير TP53. هذا هو التحليل العددي (Scalar). كل جين هو رقم واحد في كل عينة.

At this stage of the path, you looked at one gene at a time: the mean expression of BRCA1, the variance of TP53. This is Scalar analysis. Each gene is a single number per sample.

في الجزء الثالث، أنت تنظر إلى جميع الجينات في وقت واحد. العينة ليست رقماً واحداً؛ هي متجه من آلاف الأرقام. والمقارنة بين عينتين ليست مقارنة بين رقمين؛ هي مقارنة

بين متجهين باستخدام المسافة الإقليدية أو تشابه جيب التمام.

In Part Three, you look at all genes simultaneously. A sample is not a single number; it is a Vector of thousands of numbers. Comparing two samples is not comparing two numbers; it is comparing two Vectors using Euclidean distance or cosine similarity.

التطبيق على مشروعنا:

Application to Our Project:

في الباب السادس، كنا نستعلم عن العينات باستخدام SELECT * FROM samples WHERE patient_id = 5. هذا يعيد لنا صفوفاً. في الباب السابع، نأخذ هذه الصفوف ونحولها إلى متجهات، ثم نطبق عليها عمليات رياضية لا يمكن تطبيقها على الصفوف الفردية.

In Unit Six, we queried samples using SELECT * FROM samples WHERE patient_id = 5. This returned rows. In Unit Seven, we take these rows and convert them into vectors, then apply mathematical operations that cannot be applied to individual rows.

مثال:

Example:

S001 (مريض) 1200 450 2100 800

S002 (مريض) 980 520 1800 750

العينة (Sample) MYC EGFR TP53 BRCA1

S003 (سليم) 100 200 80 50

في الإحصاء القياسي (الجزء الثاني)، قد تحسب متوسط

BRCA1 في المرضى: (1200 +980)/2 = 1090.

في الجبر الخطي (الجزء الثالث)، تنظر إلى S001 و S002 كمتجهين في فضاء رباعي الأبعاد، وتحسب المسافة بينهما لتقرر مدى تشابههما، ثم تنظر إلى المسافة بينهما وبين S003.

In scalar statistics (Part Two), you might calculate the average BRCA1 in patients: (1200+980)/2 = 1090.
In linear algebra (Part Three), you look at S001 and S002 as vectors in a four-dimensional space, calculate the distance between them to decide how similar they are, and then look at the distance between them and S003.

هذا الانتقال من "رقم واحد" إلى "فضاء متجهي" هو ما يجعل PCA ممكناً (تقليل الأبعاد)، والتجميع (Clustering) ممكناً (تقسيم العينات إلى مجموعات)، وأخيراً الحوسبة الكمومية ممكنة (معالجة المتجهات في فضاء هيلبرت).

This transition from "single number" to "vector space" is what makes PCA possible (dimensionality reduction), Clustering possible (grouping samples), and finally Quantum Computing possible (processing vectors in Hilbert space).

الانتقال إلى الباب الثامن:

Transition to Unit Eight:

الآن أصبحت البيانات متجهات ومصفوفات، وأصبحت التحويلات خطية، وأصبحت القيم الذاتية مألوفة. هذا هو الأساس الذي سيبني عليه الباب الثامن: الحوسبة العلمية (Scientific Computing). سنتعلم كيف نمثل هذه المتجهات والمصفوفات بشكل عددي (Numerical Representation) على الحاسوب، ونتعامل مع أخطاء الفاصلة العائمة (Floating Point Errors)، والاستقرار العددي (Numerical Stability)، ثم ننتقل إلى

مفهوم التنسور (Tensor) الذي هو تعميم للمصفوفة إلى أبعاد أعلى، وهو البوابة النهائية إلى الحوسبة الكمومية.

Now data is vectors and matrices, transformations are linear, and eigenvalues are familiar. This is the foundation upon which Unit Eight will be built: Scientific Computing. We will learn how to represent these vectors and matrices numerically on the computer, handle Floating Point Errors, Numerical Stability, and then move to the concept of Tensor, which is a generalization of the matrix to higher dimensions, and the final gateway to Quantum Computing.

الباب الثامن: من الجبر الخطي إلى الحوسبة العلمية — تمثيل الرياضيات في الآلة

Unit Eight: From Linear Algebra to Scientific Computing — Representing Mathematics in the Machine

الغرض من هذا الباب:

تحويل المفاهيم الرياضية (المتجهات، المصفوفات، التحويلات الخطية، القيم الذاتية) من كيانات مجردة على الورق إلى كيانات رقمية يمكن للحاسوب معالجتها بكفاءة ودقة. هذا الباب هو الجسر الذي يربط بين الرياضيات النظرية والحوسبة الفعلية. سنتعلم كيف تُخزن الأعداد في الذاكرة (Floating Point)، ولماذا بعض العمليات الحسابية تفقد دقتها،

وكيف نكتب كوداً مستقراً عددياً، وأخيراً نصل إلى مفهوم التنسور (Tensor) الذي هو تعميم المصفوفة إلى أبعاد أعلى، وهو البوابة المباشرة إلى الحوسبة الكمومية حيث كل شيء هو تنسور في فضاء هيلبرت.

Purpose of this unit:

To transform mathematical concepts (vectors, matrices, linear transformations, eigenvalues) from abstract entities on paper into digital entities that computers can process efficiently and accurately. This unit is the bridge between theoretical mathematics and actual computing. We will learn how numbers are stored in memory (Floating Point), why some calculations lose precision, how to write numerically stable code, and finally reach the concept of Tensor, which is a generalization of the matrix to higher dimensions, and the direct gateway to Quantum Computing where everything is a tensor in Hilbert space.

الفصل الثامن والثلاثون: التمثيل العددي (**Numerical Representation**) — كيف ترى الآلة الأعداد؟

عندما تنتقل من SQL إلى المصفوفات، لا تنتقل البيانات وحدها؛ ينتقل معها افتراض عن النوع والحجم والوحدة والدقة. اختيار float32 بدل float64، مثلًا، يغيّر الذاكرة المتاحة والدقة المتوقعة، بينما خلط أعداد صحيحة مع عشرية قد يغير نوع العملية الناتجة. لذلك يجب أن يصبح dtype جزءًا من تصميم التجربة الحسابية، لا تفصيلًا يترك للمكتبة كي تختاره ضمنيًا. وفي البيانات الجينومية، قد يكون الفرق بين قيمة محسوبة وقيمة مقاسة مهمًا بقدر أهمية الرقم نفسه.

When you move from SQL to arrays, you are not moving data alone; you are also carrying assumptions about type, size, units, and precision. Choosing float32 rather than float64, for example, changes both memory use and numerical accuracy, while mixing integer and floating-point values can change the resulting computation type. The dtype should therefore be part of the computational experiment design rather than an implicit library choice. In genomic data, the distinction between a measured quantity and a computed quantity can be as important as the numerical value itself.

Chapter Thirty-Eight: Numerical Representation — How Does the Machine See Numbers؟

الشكل 39 — الخريطة المفاهيمية للفصل

Figure 39 — Conceptual map of the chapter

المعرفة التي بُنيت في الجزء الثاني تقودنا هنا إلى أن، استخدمت الإحصاء والمصفوفات في تحليل البيانات؛ الآن ستبني فوق ذلك لغة رياضية للحسابات العلمية.

الشكل 39 — سير العمل الهندسي للفصل

Figure 39 — Engineering workflow of the chapter

As you move from the previous part into this chapter, you used statistics and matrices to analyze data; now you will build on them as a mathematical language for scientific computation.

في الرياضيات على الورق، الأعداد حقيقية (Real Numbers) لها دقة لا نهائية. على الحاسوب، المساحة محدودة. لذلك، نمثل الأعداد بصيغة الفاصلة العائمة (Floating Point) التي تقرب

الأعداد إلى أقرب قيمة ممكنة ضمن عدد محدد من البتات. هذا التقريب هو مصدر معظم الأخطاء في الحوسبة العلمية.

In mathematics on paper, numbers are Real Numbers with infinite precision. On a computer, space is limited. Therefore, we represent numbers in Floating Point format, which approximates numbers to the nearest possible value within a fixed number of bits. This approximation is the source of most errors in scientific computing.

38.1 تمثيل الأعداد الصحيحة والأعداد الحقيقية

38.1 Integer and Floating-Point Representation.

Python

```
import sys
import struct


class NumericalRepresentation:
    """شرح تمثيل الأعداد في الحاسوب.
    Explanation of numerical representation in computers.
    """
```

```
@staticmethod
def explain_integer():
    """شرح تمثيل الأعداد الصحيحة.
    Explanation of integer representation.
    """
    print("=" * 60)
    print("تمثيل الأعداد الصحيحة (Integer Representation)")
    print("=" * 60)

    # حجم العدد الصحيح في البايتات
    int_size = sys.getsizeof(0)
    print(f"حجم العدد الصحيح في Python: {int_size} بايت")

    # نطاق العدد الصحيح 32-bit
    print(f"نطاق int 32-bit: -2,147,483,648 إلى 2,147,483,647")
    print(f"نطاق int 64-bit: -9.22e18 إلى 9.22e18")

    # مثال: تجاوز السعة (Overflow) في اللغات منخفضة المستوى
    print("\n⚠️ في Python، الأعداد الصحيحة غير محدودة (Big Integers).")
    print("في C/C++/Java، تجاوز السعة يحدث عند تخطي النطاق.")

@staticmethod
def explain_floating_point():
    """شرح تمثيل أعداد الفاصلة العائمة (IEEE 754).
    Explanation of floating-point representation (IEEE 754).
    """
    print("=" * 60)
    print("تمثيل أعداد الفاصلة العائمة (IEEE 754)")
    print("=" * 60)

    # تمثيل الرقم 0.1 في binary (IEEE 754 double precision)
```

```
    import struct
    binary_representation = struct.pack('d', 0.1)
    hex_representation = binary_representation.hex()
    print(f"0.1 في تمثيل IEEE 754 Double (hex): {hex_representation}")

    # تحليل البتات
    print("\nتتكون قيمة الفاصلة العائمة من ثلاثة أجزاء:")
    print("  1. إشارة (Sign): 1 بت")
    print("  2. الأس (Exponent): 11 بت (للـ Double)")
    print("  3. العشير (Mantissa/Significand): 52 بت (للـ Double)")
    print("\nالصيغة: (-1)^sign × (1.mantissa) × 2^(exponent - bias)")

    # عرض دقة مختلفة
    print(f"\nدقة Float (32-bit): ~7 خانات عشرية")
    print(f"دقة Double (64-bit): ~15-16 خانة عشرية")

@staticmethod
def explain_precision_limits():
    """شرح حدود الدقة.
    Explanation of precision limits.
    """
    print("=" * 60)
    print("حدود الدقة في الفاصلة العائمة")
    print("=" * 60)

    # أصغر عدد موجب يمكن تمثيله
    smallest_positive = sys.float_info.min
    print(f"أصغر عدد موجب (min): {smallest_positive:.4e}")

    # أكبر عدد يمكن تمثيله
    largest = sys.float_info.max
    print(f"أكبر عدد (max): {largest:.4e}")
```

```
 # الدقة الآلية (Machine Epsilon)
 epsilon = sys.float_info.epsilon
 print(f"الدقة الآلية (epsilon): {epsilon:.4e}")
 print(f" 1 يعني أن + epsilon ≠ 1، 1 لكن + epsilon/2 = 1")

# مثال: عرض تمثيل الأعداد
# Example: Display numerical representation
# NumericalRepresentation.explain_floating_point()
```

38.2 لماذا 0.1 + 0.2 ≠ 0.3 في الحاسوب؟

38.2 Why 0.1 + 0.2 ≠ 0.3 in a Computer؟

هذا هو المثال الكلاسيكي الذي يربك المبرمجين. السبب هو أن 0.1 و 0.2 لا يمكن تمثيلهما بدقة في النظام الثنائي (Binary)، تماماً كما لا يمكن تمثيل 1/3 بدقة في النظام العشري (0.33333...).

This is the classic example that confuses programmers. The reason is that 0.1 and 0.2 cannot be represented precisely in the binary system, just as 1/3 cannot be represented precisely in the decimal system (0.33333...).

Python

```
class FloatingPointDemo:
    """توضيح مشاكل دقة الفاصلة العائمة.
    Demonstration of floating-point precision issues.
    """

    @staticmethod
    def demonstrate_0_1_plus_0_2():
        """شرح لماذا 0.1 + 0.2 ≠ 0.3.
        Explanation of why 0.1 + 0.2 ≠ 0.3.
        """
        print("=" * 60)
        print("مشكلة الدقة: 0.1 + 0.2")
        print("=" * 60)

        a = 0.1
        b = 0.2
        c = a + b

        print(f"0.1 = {a:.20f}")
        print(f"0.2 = {b:.20f}")
        print(f"0.1 + 0.2 = {c:.20f}")
        print(f"0.3 = {0.3:.20f}")
        print(f"هل 0.1 + 0.2 == 0.3؟ {c == 0.3}")
```

```
    print("\nالسبب: 0.1 في الثنائي هو:")
    # تمثيل 0.1 في ثنائي (محاكاة)
    # 0.1 = 0.0001100110011001100110011001100110011001100110011001101...
    print("0.0001100110011001100110011001100110011001100110011001101 (ثنائي)")
    print("هذا كسر دوري لا نهائي، لذا يقطع عند حد معين.")

@staticmethod
def demonstrate_float_comparison():
    """طرق صحيحة لمقارنة أعداد الفاصلة العائمة.
    Correct ways to compare floating-point numbers.
    """
    print("\n" + "=" * 60)
    print("مقارنة أعداد الفاصلة العائمة بشكل صحيح")
    print("=" * 60)

    a = 0.1 + 0.2
    b = 0.3

    # الطريقة الخاطئة
    print(f"a == b: {a == b} (خطأ)")

    # الطريقة الصحيحة: استخدام تسامح (Tolerance)
    tolerance = 1e-10
    are_equal = abs(a - b) < tolerance
    print(f"abs(a - b) < 1e-10: {are_equal} (صحيح)")

    # طريقة أخرى: استخدام math.isclose (Python 3.5+)
    import math
    print(f"math.isclose(a, b): {math.isclose(a, b)} (صحيح)")

@staticmethod
def demonstrate_catastrophic_cancellation():
```

```
        """شرح الإلغاء الكارثي (Catastrophic Cancellation).
        Explanation of catastrophic cancellation.
        """
        print("\n" + "=" * 60)
        print("الإلغاء الكارثي (Catastrophic Cancellation)")
        print("=" * 60)

        # مثال: طرح رقمين متقاربين جدا
        x = 1.0000000001
        y = 1.0000000000
        diff = x - y

        print(f"x = {x:.20f}")
        print(f"y = {y:.20f}")
        print(f"x - y = {diff:.20f}")

        print("\n⚠️ عندما نطرح رقمين متقاربين جداً، نفقد دقة كبيرة.")
        print("النتيجة تعتمد على الأرقام الأخيرة غير الدقيقة في التمثيل الثنائي.")
        print("هذا يسمى 'الإلغاء الكارثي'.")

# تشغيل العرض
# FloatingPointDemo.demonstrate_0_1_plus_0_2()
# FloatingPointDemo.demonstrate_float_comparison()
# FloatingPointDemo.demonstrate_catastrophic_cancellation()
```

الفصل التاسع والثلاثون: الفاصلة العائمة (**Floating Point**) — الدقة والخطأ في العمليات الحسابية

Chapter Thirty-Nine: Floating Point — Precision and Error in Calculations

الشكل 40 — الخريطة المفاهيمية للفصل

Figure 40 — Conceptual map of the chapter

من الأدوات التي اكتسبتها في الجزء الثاني، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

الشكل 40 — سير العمل الهندسي للفصل

Figure 40 — Engineering workflow of the chapter

The knowledge built in Part Two leads directly here because you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

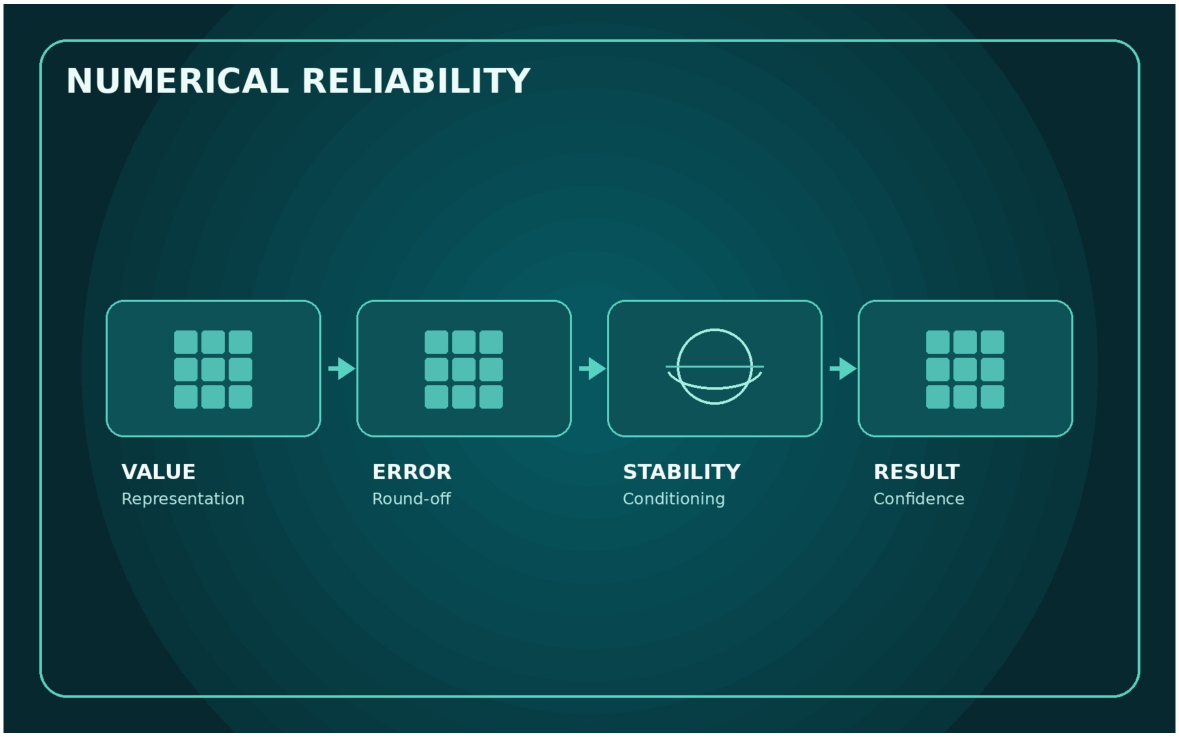


Figure — NUMERICAL RELIABILITY

في هذا الفصل، نتعمق في كيفية عمل الفاصلة العائمة، وكيف تتراكم الأخطاء عبر العمليات المتتالية، وكيف يمكننا قياس وتقدير هذه الأخطاء.

In this chapter, we delve into how floating point works, how errors accumulate across successive operations, and how we can measure and estimate these errors.

39.1 أنواع أخطاء الفاصلة العائمة

39.1 Types of Floating-Point Errors.

خطأ التقريب (Rounding Error) الفرق بين القيمة الحقيقية والقيمة المخزنة. 0.1 مخزنة كـ 0.1000000000000000555

خطأ القطع (Truncation Error) قطع عدد لا نهائي من الخانات. π مخزنة كـ 3.141592653589793 بدلاً من القيمة اللانهائية

نوع الخطأ (Error Type) الوصف (Description) المثال (Example)

خطأ التراكم (Accumulation Error) تراكم أخطاء التقريب عبر عمليات متعددة. جمع 1,000,000 رقم صغير جداً

الإلغاء الكارثي (Catastrophic Cancellation) طرح رقمين متقاربين يفقد الدقة. 1.0 - 1.0000000001

تمثيل الأخطاء في OOP:

Error Representation in OOP:

Python

```
from dataclasses import dataclass
from typing import Tuple
import math

@dataclass
class NumericalError:
    """تمثيل الخطأ العددي في عملية حسابية.
    Numerical error representation in a calculation.
    """
    true_value: float
    computed_value: float
    absolute_error: float
    relative_error: float

    @classmethod
    def from_calculation(cls, true_value: float, computed_value: float) -> 'NumericalError':
        """حساب الأخطاء من القيم الحقيقية والمحسوبة.
        Calculate errors from true and computed values.
        """
        abs_error = abs(true_value - computed_value)
        rel_error = abs_error / abs(true_value) if true_value != 0 else float('inf')
        return cls(
            true_value=true_value,
            computed_value=computed_value,
            absolute_error=abs_error,
```

```
            relative_error=rel_error
        )

    def __repr__(self) -> str:
        return (
            f"NumericalError(\n"
            f"  true={self.true_value:.10f},\n"
            f"  computed={self.computed_value:.10f},\n"
            f"  absolute_error={self.absolute_error:.4e},\n"
            f"  relative_error={self.relative_error:.4e}\n"
            f")"
        )


class FloatingPointAnalysis:
    """تحليل دقة العمليات الحسابية.
    Analysis of computational precision.
    """

    @staticmethod
    def analyze_accumulation(n: int = 1000000) -> NumericalError:
        """تحليل تراكم الأخطاء عند جمع أعداد صغيرة.
        Analyze error accumulation when summing small numbers.
        """
        # القيمة الحقيقية: جمع 1/n لـ n مرة = 1
        true_value = 1.0

        # الطريقة الأولى: جمع مباشر (تراكم الأخطاء)
        computed_direct = 0.0
        for _ in range(n):
            computed_direct += 1.0 / n
```

```
    # الطريقة الثانية: جمع باستخدام Kahan (تقليل الأخطاء)
    computed_kahan = 0.0
    compensation = 0.0
    for _ in range(n):
        y = (1.0 / n) - compensation
        t = computed_kahan + y
        compensation = (t - computed_kahan) - y
        computed_kahan = t

    print("=" * 60)
    print(f"تراكم الأخطاء عند جمع {n} مرة من 1/{n}")
    print("=" * 60)
    print(f"القيمة الحقيقية: {true_value:.10f}")
    print(f"الجمع المباشر: {computed_direct:.10f} (خطأ: {abs(true_value - computed_direct):.4e})")
    print(f"جمع Kahan: {computed_kahan:.10f} (خطأ: {abs(true_value - computed_kahan):.4e})")

    return NumericalError.from_calculation(true_value, computed_direct)

@staticmethod
def kahan_sum(values: list) -> float:
    """
    خوارزمية جمع Kahan (تعويض الخطأ) لتقليل تراكم الأخطاء.
    Kahan summation algorithm (error compensation) to reduce error accumulation.
    """
    total = 0.0
    compensation = 0.0

    for value in values:
        y = value - compensation
        t = total + y
        compensation = (t - total) - y
        total = t
```

```
    return total
```

39.2 تمثيل المصفوفات بشكل عددي (مستودع متجهات رقمي)

39.2 Numerical Representation of Matrices (Numeric Vector Repository).

الآن نربط التمثيل العددي بقاعدة البيانات: عند استرجاع متجهات التعبير، نقوم بتحويلها إلى numpy arrays للاستفادة من العمليات المحسّنة، مع الحفاظ على دقة Double (64-bit) افتراضياً.

Now we link numerical representation to the database: when retrieving expression vectors, we convert them to numpy arrays to leverage optimized operations, maintaining Double precision (64-bit) by default.

Python

```python
import numpy as np
from typing import List, Optional, Dict

class NumericVectorRepository:
    """
    مستودع يستخدم numpy لتمثيل المتجهات والمصفوفات عدديًا.
    Repository using numpy for numerical vector and matrix representation.
    """
    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path
        self.dtype = np.float64  # دقة مزدوجة (Double Precision)

    def get_sample_vector_numpy(self, sample_id: int, gene_list: List[str]) -> Optional[np.ndarray]:
        """
        استرجاع متجه التعبير كـ numpy array بدقة مزدوجة.
        Retrieve expression vector as a numpy array with double precision.
        """
        if not gene_list:
            return None

        with sqlite3.connect(self.db_path) as conn:
            conn.row_factory = sqlite3.Row
            cursor = conn.cursor()

            placeholders = ','.join(['?'] * len(gene_list))
            query = f"""
                SELECT gene_symbol, expression_count
                FROM read_counts
                WHERE sample_id = ? AND gene_symbol IN ({placeholders})
            """
            params = [sample_id] + gene_list
            rows = cursor.execute(query, params).fetchall()
```

```
        # بناء قاموس القيم
        value_map = {row["gene_symbol"]: row["expression_count"] for row in rows}

        # تحويل إلى numpy array بدقة مزدوجة
        components = np.array(
            [float(value_map.get(gene, 0.0)) for gene in gene_list],
            dtype=self.dtype
        )
        return components

    def get_expression_matrix_numpy(self, sample_ids: List[int], gene_list: List[str]) ->
Optional[np.ndarray]:
        """
        بناء مصفوفة التعبير كـ numpy array (عينات × جينات).
        Build expression matrix as a numpy array (samples × genes).
        """
        if not sample_ids or not gene_list:
            return None

        matrix_rows = []
        for sid in sample_ids:
            vec = self.get_sample_vector_numpy(sid, gene_list)
            if vec is not None:
                matrix_rows.append(vec)

        if not matrix_rows:
            return None

        # تحويل إلى مصفوفة numpy ثنائية الأبعاد
        return np.vstack(matrix_rows).astype(self.dtype)
```

```
    def compute_sample_similarity(self, sample_id1: int, sample_id2: int, gene_list: List[str]) ->
Dict[str, float]:
        """
        حساب مقاييس التشابه بين عينتين باستخدام numpy.
        Compute similarity measures between two samples using numpy.
        """
        v1 = self.get_sample_vector_numpy(sample_id1, gene_list)
        v2 = self.get_sample_vector_numpy(sample_id2, gene_list)

        if v1 is None or v2 is None:
            return {}

        # المسافة الإقليدية
        euclidean = np.linalg.norm(v1 - v2)

        # تشابه جيب التمام
        dot_product = np.dot(v1, v2)
        norm_product = np.linalg.norm(v1) * np.linalg.norm(v2)
        cosine = dot_product / norm_product if norm_product > 0 else 0.0

        # معامل ارتباط بيرسون
        corr = np.corrcoef(v1, v2)[0, 1]

        return {
            "euclidean_distance": float(euclidean),
            "cosine_similarity": float(cosine),
            "pearson_correlation": float(corr)
        }
```

الفصل الأربعون: الدقة والخطأ (Precision and Error) — متى تثق بالنتيجة؟

Chapter Forty: Precision and Error — When to Trust the Result?

الشكل 41 — الخريطة المفاهيمية للفصل

Figure 41 — Conceptual map of the chapter

الشكل 41 — سير العمل الهندسي للفصل

Figure 41 — Engineering workflow of the chapter

بعد أن أصبحت الأساسيات السابقة مستقرة، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

Among the tools you gained in Part Two, you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

40.5 تقدير الخطأ في الحسابات البيولوجية — من القيمة الخام إلى الثقة العددية

40.5 Estimating Error in Biological Computation — From Raw Values to Numerical Confidence.

الحساب العددي لا يتعامل مع الأعداد الحقيقية المثالية؛ تمثيل الأعداد في الحاسوب محدود. لذلك قد لا يساوي 0.1 + 0.2 القيمة النصية 0.3 عند اختبار المساواة المباشرة. هذه ليست مشكلة في الرياضيات، بل نتيجة لطريقة تمثيل الأعداد ذات الفاصلة العائمة. توثيق Python الرسمي يشرح هذه الظاهرة بوضوح ويبين أن كثيرًا من الكسور العشرية لا يمكن تمثيلها ثنائيًا بدقة تامة.

Numerical computation does not operate on ideal real numbers; computer representations are finite. Therefore 0.1 + 0.2 may not equal the textual value 0.3 under direct equality. This is not a failure of mathematics, but a consequence of binary floating-point representation. Python's official documentation explains that many decimal fractions cannot be represented exactly in binary floating point.

Python

```
import math
```

```
x = 0.1 + 0.2
print(x)
print(math.isclose(x, 0.3, rel_tol=1e-12, abs_tol=1e-12))
```

في البيانات الجينومية، يصبح هذا أكثر أهمية عند تكرار العمليات، أو مقارنة قيم قريبة جدًا، أو حساب فروق صغيرة بين نماذج. لذلك ستتعلم التمييز بين الخطأ النسبي والخطأ المطلق، وبين الدقة العددية وعدم اليقين الإحصائي؛ فهما ليسا الشيء نفسه.

In genomic data, this becomes more important when operations are repeated, when values are extremely close, or when small model differences matter. You therefore need to distinguish relative error and absolute error from statistical uncertainty; numerical precision and statistical uncertainty are not the same thing.

في الحوسبة العلمية، النتيجة ليست مجرد رقم؛ هي رقم مع هامش خطأ. فهم هذا الهامش هو ما يفصل بين المهندس

الجيد والمبرمج العادي. في هذا الفصل، نتعلم كيف نقدر عدم اليقين في حساباتنا، وكيف نبلغ عنه في التقارير العلمية.

In scientific computing, a result is not just a number; it is a number with an Error Margin. Understanding this margin is what separates a good engineer from an average programmer. In this chapter, we learn how to estimate uncertainty in our calculations, and how to report it in scientific reports.

40.1 تقدير الخطأ في العمليات الحسابية

40.1 Estimating Error in Calculations.

Python

```
class ErrorEstimation:
    """تقدير الأخطاء في العمليات الحسابية.
    Error estimation in calculations.
    """

    @staticmethod
    def relative_error(computed: float, true_value: float) -> float:
        """حساب الخطأ النسبي.
        Calculate relative error.
```

```
    """
    if true_value == 0:
        return float('inf')
    return abs((computed - true_value) / true_value)

@staticmethod
def absolute_error(computed: float, true_value: float) -> float:
    """حساب الخطأ المطلق.
    Calculate absolute error.
    """
    return abs(computed - true_value)

@staticmethod
def propagate_uncertainty_simple(values: List[float], uncertainties: List[float]) -> tuple:
    """
    نشر عدم اليقين لجمع بسيط (الخطأ التربيعي الجذري).
    Propagate uncertainty for simple addition (root-sum-square).
    """
    if len(values) != len(uncertainties):
        raise ValueError("عدد القيم يجب أن يساوي عدد حالات عدم اليقين")

    # القيمة المجمعة
    combined_value = sum(values)

    # عدم اليقين المجمع (RSS: Root Sum of Squares)
    combined_uncertainty = math.sqrt(sum(u**2 for u in uncertainties))

    return combined_value, combined_uncertainty

@staticmethod
def confidence_interval_mean(samples: List[float], confidence_level: float = 0.95) -> tuple:
    """
```

```
حساب فاصل الثقة للمتوسط باستخدام توزيع t.
Calculate confidence interval for the mean using t-distribution.
"""
try:
    from scipy import stats
    n = len(samples)
    mean = sum(samples) / n
    std = math.sqrt(sum((x - mean)**2 for x in samples) / (n - 1))
    t_critical = stats.t.ppf((1 + confidence_level) / 2, n - 1)
    margin = t_critical * std / math.sqrt(n)
    return mean - margin, mean + margin
except ImportError:
    # إذا لم يكن scipy متاحا تقريب باستخدام 1.96 (لتوزيع طبيعي)
    n = len(samples)
    mean = sum(samples) / n
    std = math.sqrt(sum((x - mean)**2 for x in samples) / (n - 1))
    margin = 1.96 * std / math.sqrt(n)
    return mean - margin, mean + margin
```

40.2 الدقة في التحليل الإحصائي (ربط مع الجزء الثاني)

40.2 Precision in Statistical Analysis (Linking to Part Two).

ما تعلّمته سابقًا عن البيانات والبرمجة يصبح هنا نقطة انطلاق، تعلمت حساب قيمة p وفحص الفرضيات. الآن نضيف طبقة من الدقة العددي: هل قيمة p المحسوبة موثوقة؟ هل حجم العينة كافٍ للوصول إلى هذا المستوى من الدقة؟

Once the earlier foundations are stable, you learned to calculate p-values and test hypotheses. Now we add a layer of Numerical Precision: is the calculated p-value reliable? Is the sample size sufficient to achieve this level of precision?

Python

```
class StatisticalPrecision:
    """تحليل دقة النتائج الإحصائية.
    Precision analysis of statistical results.
    """

    @staticmethod
    def monte_carlo_p_value(
        test_statistic: float,
        data_generator: callable,
        n_simulations: int = 10000
    ) -> tuple:
        """
        تقدير قيمة p باستخدام محاكاة مونت كارلو مع تقدير الخطأ.
        Estimate p-value using Monte Carlo simulation with error estimation.
        """
        # توليد إحصائيات تحت الفرضية العدمية
        null_stats = []
        for _ in range(n_simulations):
            # توليد بيانات عشوائية تحت H₀
            simulated_data = data_generator()
            # حساب إحصائية الاختبار (هنا مثال بسيط)
            # في الواقع، هذا يعتمد على الاختبار المحدد
            # هيكل توضيحي فقط
```

```
        pass

    # حساب قيمة p
    # p_value = proportion of null_stats >= test_statistic
    # error = sqrt(p * (1-p) / n_simulations)  # خطأ معياري لنسبة

    # إرجاع قيمة p مع هامش الخطأ
    p_value = 0.05  # قيمة وهمية للتوضيح
    p_error = math.sqrt(p_value * (1 - p_value) / n_simulations)

    return p_value, p_error

@staticmethod
def minimum_sample_size(
    effect_size: float,
    power: float = 0.8,
    alpha: float = 0.05
) -> int:
    """
    تقدير الحد الأدنى لحجم العينة لاختبار t (محاكاة بسيطة).
    Estimate minimum sample size for a t-test (simple simulation).
    """
    # هذه محاكاة تعليمية، في الواقع نستخدم pow
    # This is an educational simulation; in practice, use power analysis libraries
    # n *2 ( z_alpha + z_beta)^2 / effect_size^2
    import math
    from scipy import stats

    z_alpha = stats.norm.ppf(1 - alpha/2)
    z_beta = stats.norm.ppf(power)

    n = 2 * ((z_alpha + z_beta)**2) / (effect_size**2)
```

```
    return int(math.ceil(n))
```

الفصل الحادي والأربعون: الاستقرار العددي (**Numerical Stability**) — كتابة كود لا ينهار

Chapter Forty-One: Numerical Stability — Writing Code That Does Not Collapse

الاستقرار العددي يختلف عن الدقة العددية. قد تستخدم نوعًا عالي الدقة ومع ذلك تحصل على خوارزمية حساسة للأخطاء، لأن الخطأ الصغير في المدخل قد يتضخم أثناء العمليات. لهذا نحتاج إلى التفكير في conditioning، وحجم القيم، وتراكم الخطأ، واختيار الخوارزمية، وليس فقط عدد المنازل العشرية. في النظام العلمي القابل للتدقيق يجب أن تسجل أيضًا الاختبارات التي استخدمتها لتحديد أن النتيجة مستقرة ضمن نطاق الاستخدام المقصود.

Numerical stability is different from numerical precision. You can use a high-precision type and still have an algorithm that is highly sensitive to small perturbations because input errors may be amplified by the computation. This is why conditioning, scale, error accumulation, algorithm choice, and not merely decimal places must be considered. In an auditable scientific system, you should also record the tests used to establish that a result is stable within its intended operating range.

الشكل 42 — الخريطة المفاهيمية للفصل

Figure 42 — Conceptual map of the chapter

الخطوة التالية في هذا المسار تبدأ مما أُنجز في الجزء الثاني، استخدمت الإحصاء والمصفوفات في تحليل البيانات؛ الآن ستبني فوق ذلك لغة رياضية للحسابات العلمية.

الشكل 42 — سير العمل الهندسي للفصل

Figure 42 — Engineering workflow of the chapter

What you learned earlier about data and programming now becomes the starting point, you used statistics and matrices to analyze data; now you will build on them as a mathematical language for scientific computation.

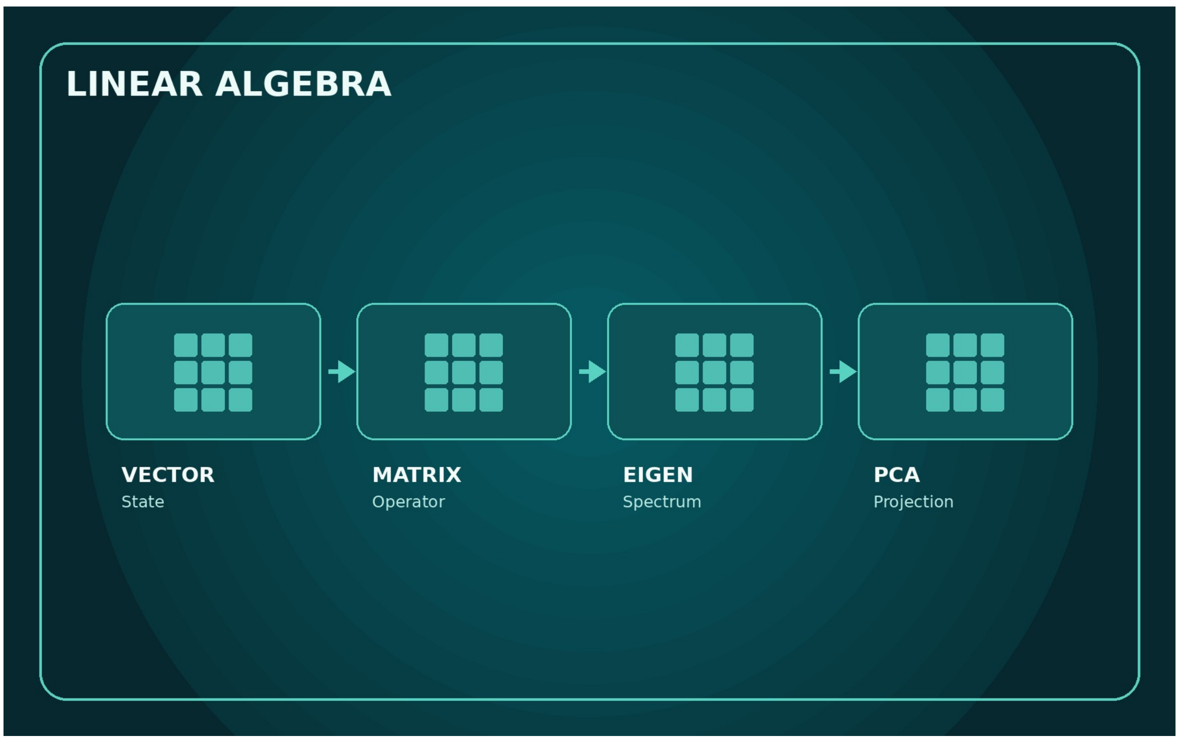


Figure — LINEAR ALGEBRA

بعض الخوارزميات تكون "غير مستقرة عددياً"، مما يعني أن خطأ صغير في المدخلات يتضخم ليصبح خطأ كبيراً في

المخرجات. فهم الاستقرار العددي هو ما يمنع الكود من الانهيار عند التعامل مع بيانات حقيقية.

Some algorithms are "numerically unstable," meaning a small error in the input is amplified into a large error in the output. Understanding numerical stability is what prevents code from collapsing when handling real data.

41.1 أمثلة على عدم الاستقرار العددي

41.1 Examples of Numerical Instability.

Python

```
class NumericalStability:
    """توضيح الاستقرار العددي وعدم الاستقرار.
    Demonstration of numerical stability and instability.
    """

    @staticmethod
    def unstable_quadratic(a: float, b: float, c: float) -> tuple:
        """
        حل المعادلة التربيعية باستخدام الصيغة التقليدية (غير مستقرة لبعض القيم).
        Solve quadratic equation using the traditional formula (unstable for some values).
```

```
    """
    import math
    discriminant = b**2 - 4*a*c

    if discriminant < 0:
        return None

    sqrt_disc = math.sqrt(discriminant)
    x1 = (-b + sqrt_disc) / (2*a)
    x2 = (-b - sqrt_disc) / (2*a)
    return x1, x2

@staticmethod
def stable_quadratic(a: float, b: float, c: float) -> tuple:
    """
    حل المعادلة التربيعية باستخدام صيغة مستقرة عدديا.
    Solve quadratic equation using a numerically stable formula.
    """
    import math
    discriminant = b**2 - 4*a*c

    if discriminant < 0:
        return None

    sqrt_disc = math.sqrt(discriminant)

    # اختيار الإشارة لتجنب الإلغاء الكارثي
    if b >= 0:
        q = -0.5 * (b + sqrt_disc)
    else:
        q = -0.5 * (b - sqrt_disc)
```

```
        x1 = q / a
        x2 = c / q

        return x1, x2

    @staticmethod
    def demonstrate_instability():
        """عرض الفرق بين الصيغتين.
        Demonstrate the difference between the two formulas.
        """
        print("=" * 60)
        print("الاستقرار العددي في حل المعادلة التربيعية")
        print("=" * 60)

        # حالة تسبب عدم استقرار: a صغير، b و c كبيران
        a = 1e-20
        b = 1e20
        c = 1e20

        print(f"a = {a:.2e}, b = {b:.2e}, c = {c:.2e}")

        unstable = NumericalStability.unstable_quadratic(a, b, c)
        stable = NumericalStability.stable_quadratic(a, b, c)

        if unstable:
            print(f"الصيغة غير المستقرة: x1 = {unstable[0]:.2e}, x2 = {unstable[1]:.2e}")
        if stable:
            print(f"الصيغة المستقرة: x1 = {stable[0]:.2e}, x2 = {stable[1]:.2e}")

    @staticmethod
    def softmax_stable(logits: List[float]) -> List[float]:
        """
```

```
    دالة Softmax (تطرح القيمة القصوى لتجنب التفيض) مستقرة عددياً.
    Numerically stable Softmax function (subtracts max to avoid overflow).
    """
    import math

    # إيجاد القيمة القصوى
    max_logit = max(logits)

    # طرح القيمة القصوى من جميع العناصر (تجعل القيم <= 0)
    shifted = [x - max_logit for x in logits]

    # حساب الأس (الآن القيم <= 0، لذا لا يوجد تفيض)
    exp_vals = [math.exp(x) for x in shifted]

    # التطبيع
    sum_exp = sum(exp_vals)
    return [e / sum_exp for e in exp_vals]

@staticmethod
def log_sum_exp_stable(logits: List[float]) -> float:
    """
    حساب log(sum(exp(x))) بشكل مستقر عدديا.
    Calculate log(sum(exp(x))) in a numerically stable way.
    """
    import math

    if not logits:
        return float('-inf')

    max_logit = max(logits)

    # log(sum(exp(x - max)))
```

```
    sum_exp = sum(math.exp(x - max_logit) for x in logits)
    return max_logit + math.log(sum_exp)
```

الفصل الثاني والأربعون: المتجهات والمصفوفات حسابياً — NumPy كأساس للحوسبة العلمية

Chapter Forty-Two: Vectors and Matrices Computationally — NumPy as the Scientific Computing Foundation

الشكل 43 — الخريطة المفاهيمية للفصل

Figure 43 — Conceptual map of the chapter

عندما نصل إلى هذه الطبقة الهندسية، استخدمت الإحصاء والمصفوفات في تحليل البيانات؛ الآن ستبني فوق ذلك لغة رياضية للحسابات العلمية.

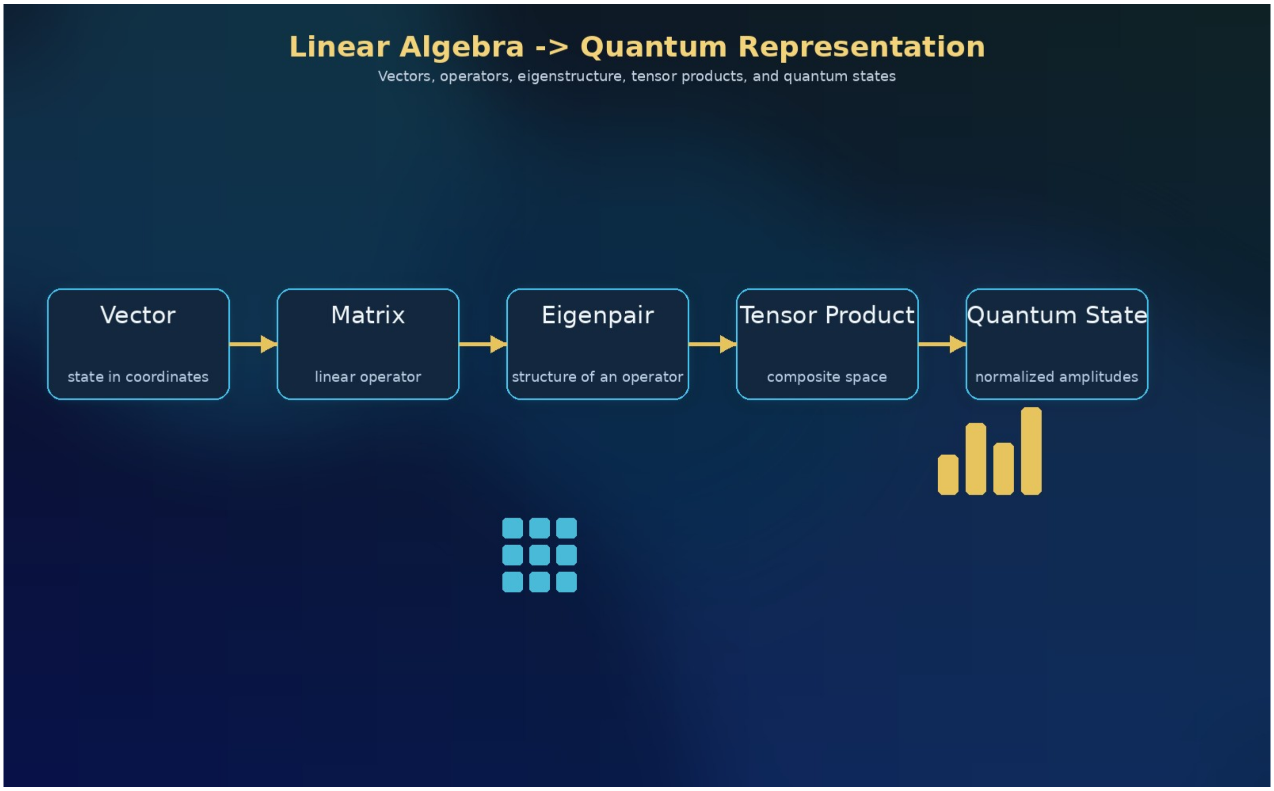
الشكل 43 — سير العمل الهندسي للفصل

Figure 43 — Engineering workflow of the chapter

42.5 مصفوفات الاستبدال كبوابة إلى الجبر الخطي

The next engineering step grows out of the work completed in Part Two, you used statistics and matrices to analyze data; now you will build on them as a mathematical language for scientific computation.

42.5 Substitution Matrices as a Gateway to Linear Algebra.



المصفوفة البيولوجية تصبح نقطة عبور إلى المتجهات والتحويلات والمشغلات الخطية.

BLOSUM62 ليست مجرد جدول درجات محفوظ. هي مصفوفة عددية يمكن تمثيلها في NumPy وتحليلها كجسم رياضي. لكن لا تخلط بين استخدامها كمصفوفة درجات في المحاذاة وبين اعتبارها تلقائيًا مؤثرًا فيزيائيًا أو مصفوفة احتمالية. وظيفتها الأصلية في BLAST هي إعطاء درجات لمقابلة أزواج من

الأحماض الأمينية، وتوثيق NCBI يصف BLOSUM62 بأنها

مصفوفة شائعة الاستخدام في BLASTP.

A biological matrix becomes a bridge to vectors, transformations, and linear operators.

BLOSUM62 is not merely a memorized scoring table. It is a numerical matrix that can be represented in NumPy and studied as a mathematical object. However, do not confuse its use as an alignment scoring matrix with automatically treating it as a physical operator or a probability matrix. Its original role in BLAST is to assign scores to amino-acid pairings, and NCBI documents BLOSUM62 as a commonly used matrix in BLASTP.

Python

```
import numpy as np


 #  A small symmetric example, not the full BLOSUM62 matrix.
S = np.array([
    [4, -1, -2],
    [-1, 5,  0],
    [-2,  0, 6]
], dtype=float)


eigenvalues, eigenvectors = np.linalg.eigh(S)


print("Eigenvalues:", eigenvalues)
print("Eigenvectors:")
```

```
print(eigenvectors)
```

هذا المثال يفتح باب eigenvalues وeigenvectors بصورة سليمة: المصفوفة تصبح مدخلًا إلى أدوات الجبر الخطي، ثم يمكن استخدام هذه الأدوات في PCA وSVD وغيرها. توثيق NumPy يوفّر مباشرة eig وeigh لحساب القيم والمتجهات الذاتية، كما يوفّر SVD وحساب الرتب والمعايير.

This example opens the door to eigenvalues and eigenvectors in a sound way: the matrix becomes an input to linear-algebra tools, which can then be used in PCA, SVD, and related methods. NumPy provides eig and eigh for eigenvalue and eigenvector computation, together with SVD, rank, and norm operations.

NumPy هو المكتبة الأساسية للحوسبة العلمية في Python. توفر عمليات مصفوفة محسّنة (مكتوبة بلغة C و Fortran)، مما يجعلها أسرع بكثير من الحلقات (Loops) في Python الخام. في

هذا الفصل، نستبدل تطبيقات Vector و Matrix اليدوية بـ numpy مع الحفاظ على واجهة OOP.

NumPy is the foundational library for scientific computing in Python. It provides optimized matrix operations (written in C and Fortran), making them much faster than loops in pure Python. In this chapter, we replace our manual Vector and Matrix implementations with numpy while maintaining an OOP interface.

42.1 واجهة OOP لـ numpy Vector و Matrix

42.1 OOP Interface for numpy Vector and Matrix.

Python

```
import numpy as np
from typing import List, Union, Optional


class NumpyVector:
    """متجه يعتمد على numpy array مع واجهة OOP.
    Vector based on numpy array with OOP interface.
    """
```

```
def __init__(self, data: Union[List[float], np.ndarray]):
    if isinstance(data, list):
        self._data = np.array(data, dtype=np.float64)
    else:
        self._data = data.astype(np.float64)

@property
def data(self) -> np.ndarray:
    return self._data

@property
def shape(self) -> tuple:
    return self._data.shape

def __len__(self) -> int:
    return len(self._data)

def __add__(self, other: 'NumpyVector') -> 'NumpyVector':
    if len(self) != len(other):
        raise ValueError("Vectors must have same length")
    return NumpyVector(self._data + other._data)

def __sub__(self, other: 'NumpyVector') -> 'NumpyVector':
    if len(self) != len(other):
        raise ValueError("Vectors must have same length")
    return NumpyVector(self._data - other._data)

def scale(self, scalar: float) -> 'NumpyVector':
    return NumpyVector(self._data * scalar)

def dot(self, other: 'NumpyVector') -> float:
```

```
        if len(self) != len(other):
            raise ValueError("Vectors must have same length")
        return np.dot(self._data, other._data)

    def norm(self, order: int = 2) -> float:
        return float(np.linalg.norm(self._data, ord=order))

    def cosine_similarity(self, other: 'NumpyVector') -> float:
        if len(self) != len(other):
            raise ValueError("Vectors must have same length")
        norm_product = self.norm() * other.norm()
        if norm_product == 0:
            return 0.0
        return float(self.dot(other) / norm_product)

    def to_list(self) -> List[float]:
        return self._data.tolist()

    def __repr__(self) -> str:
        return f"NumpyVector(shape={self.shape}, data={self._data[:5]}...)"


class NumpyMatrix:
    """مصفوفة تعتمد على numpy array مع واجهة OOP.
    Matrix based on numpy array with OOP interface.
    """

    def __init__(self, data: Union[List[List[float]], np.ndarray]):
        if isinstance(data, list):
            self._data = np.array(data, dtype=np.float64)
        else:
            self._data = data.astype(np.float64)
```

```
@property
def data(self) -> np.ndarray:
    return self._data

@property
def shape(self) -> tuple:
    return self._data.shape

@property
def rows(self) -> int:
    return self._data.shape[0] if len(self._data.shape) > 0 else 0

@property
def cols(self) -> int:
    return self._data.shape[1] if len(self._data.shape) > 1 else 0

def get_row(self, index: int) -> NumpyVector:
    if index >= self.rows:
        raise IndexError("Row index out of range")
    return NumpyVector(self._data[index, :])

def get_column(self, index: int) -> NumpyVector:
    if index >= self.cols:
        raise IndexError("Column index out of range")
    return NumpyVector(self._data[:, index])

def transpose(self) -> 'NumpyMatrix':
    return NumpyMatrix(self._data.T)

def add(self, other: 'NumpyMatrix') -> 'NumpyMatrix':
    if self.shape != other.shape:
```

```
        raise ValueError("Matrices must have same shape")
    return NumpyMatrix(self._data + other._data)

def multiply(self, other: 'NumpyMatrix') -> 'NumpyMatrix':
    if self.cols != other.rows:
        raise ValueError(
            f"Cannot multiply: {self.shape} with {other.shape}"
        )
    return NumpyMatrix(np.dot(self._data, other._data))

def scale(self, scalar: float) -> 'NumpyMatrix':
    return NumpyMatrix(self._data * scalar)

def mean(self, axis: Optional[int] = None) -> float:
    return float(np.mean(self._data, axis=axis))

def std(self, axis: Optional[int] = None) -> float:
    return float(np.std(self._data, axis=axis))

def to_numpy(self) -> np.ndarray:
    return self._data

def __repr__(self) -> str:
    return f"NumpyMatrix(shape={self.shape}, data=\n{self._data[:5, :5]}...)"
```

42.2 ربط Numpy بقاعدة البيانات —— مصفوفة التعبير المحسنة

42.2 Linking Numpy to the Database — Optimized Expression Matrix.

Python

```
class NumpyExpressionMatrixBuilder:
    """بناء مصفوفة التعبير باستخدام numpy لتحقيق أقصى أداء.
    Expression matrix builder using numpy for maximum performance.
    """

    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path

    def build_matrix(self, sample_ids: List[int], gene_list: List[str]) -> NumpyMatrix:
        """بناء مصفوفة التعبير كـ NumpyMatrix.
        Build expression matrix as NumpyMatrix.
        """
        if not sample_ids or not gene_list:
            return NumpyMatrix(np.array([[]], dtype=np.float64))

        # استرجاع جميع البيانات دفعة واحدة (استعلام واحد)
        with sqlite3.connect(self.db_path) as conn:
            placeholders_samples = ','.join(['?'] * len(sample_ids))
            placeholders_genes = ','.join(['?'] * len(gene_list))

            query = f"""
                SELECT sample_id, gene_symbol, expression_count
                FROM read_counts
                WHERE sample_id IN ({placeholders_samples})
                AND gene_symbol IN ({placeholders_genes})
            """
```

```
        params = sample_ids + gene_list
        rows = conn.execute(query, params).fetchall()

    # بناء قاموس: (sample_id, gene_symbol) → expression
    value_map = {}
    for row in rows:
        key = (row[0], row[1])
        value_map[key] = row[2]

    # بناء المصفوفة
    matrix_data = []
    for sid in sample_ids:
        row = []
        for gene in gene_list:
            row.append(float(value_map.get((sid, gene), 0.0)))
        matrix_data.append(row)

    return NumpyMatrix(matrix_data)

def build_matrix_with_metadata(self, sample_ids: List[int], gene_list: List[str]) -> dict:
    """بناء مصفوفة مع بيانات وصفية للعينات (لتحليل PCA والتجميع).
    Build matrix with sample metadata for PCA and clustering.
    """
    matrix = self.build_matrix(sample_ids, gene_list)

    # استرجاع بيانات وصفية للعينات
    with sqlite3.connect(self.db_path) as conn:
        placeholders = ','.join(['?'] * len(sample_ids))
        query = f"""
            SELECT sample_id, status, tissue_type, patient_id
            FROM samples
            WHERE sample_id IN ({placeholders})
```

```
    """
    rows = conn.execute(query, sample_ids).fetchall()

metadata = {}
for row in rows:
    metadata[row[0]] = {
        "status": row[1],
        "tissue_type": row[2],
        "patient_id": row[3]
    }

return {
    "matrix": matrix,
    "sample_ids": sample_ids,
    "gene_list": gene_list,
    "metadata": metadata
}
```

الفصل الثالث والأربعون: القيم الذاتية والمتجهات الذاتية (**Eigenvalues / Eigenvectors**) — الأنماط الخفية في البيانات

Chapter Forty-Three: Eigenvalues and Eigenvectors — Hidden Patterns in Data

الشكل 44 — الخريطة المفاهيمية للفصل

Figure 44 — Conceptual map of the chapter

الخبرة السابقة في Python وBiopython تمهّد مباشرة لهذه الخطوة، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

الشكل 44 — سير العمل الهندسي للفصل

Figure 44 — Engineering workflow of the chapter

When we reach this engineering layer, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

القيم الذاتية والمتجهات الذاتية هي جوهر تحليل البيانات متعددة الأبعاد. في المعلوماتية الحيوية، تستخدم في PCA (لتقليل الأبعاد)، وتحليل المكونات الرئيسية، وفهم الأنماط الخفية في التعبير الجيني.

Eigenvalues and eigenvectors are the heart of multi-dimensional data analysis. In bioinformatics, they are used in PCA (dimensionality reduction), principal component analysis, and understanding hidden patterns in gene expression.

43.1 تحليل القيم الذاتية باستخدام numpy

43.1 Eigenvalue Analysis Using numpy.

Python

```
class EigenAnalysis:
    """تحليل القيم الذاتية والمتجهات الذاتية.
    Eigenvalue and eigenvector analysis.
    """
```

```
@staticmethod
def compute_eigen(matrix: NumpyMatrix) -> dict:
    """
    حساب القيم الذاتية والمتجهات الذاتية لمصفوفة مربعة.
    Compute eigenvalues and eigenvectors of a square matrix.
    """
    if matrix.rows != matrix.cols:
        raise ValueError("المصفوفة يجب أن تكون مربعة")

    # حساب القيم الذاتية والمتجهات الذاتية
    eigenvalues, eigenvectors = np.linalg.eig(matrix.to_numpy())

    # ترتيب تنازلي حسب القيم الذاتية
    idx = np.argsort(eigenvalues)[::-1]
    eigenvalues = eigenvalues[idx]
    eigenvectors = eigenvectors[:, idx]

    return {
        "eigenvalues": eigenvalues.tolist(),
        "eigenvectors": eigenvectors.tolist(),
        "explained_variance_ratio": [
            float(ev / sum(eigenvalues)) for ev in eigenvalues
        ]
    }

@staticmethod
def pca_analysis(expression_matrix: NumpyMatrix, n_components: int = 2) -> dict:
    """
    تحليل PCA باستخدام تحليل القيم الذاتية لمصفوفة التغاير.
    PCA analysis using eigenvalue decomposition of the covariance matrix.
    """
```

```
# 1. مركزية البيانات (طرح المتوسط من كل عمود)
# 1. Center data (subtract mean from each column)
data = expression_matrix.to_numpy()
mean_centered = data - np.mean(data, axis=0)

# 2. حساب مصفوفة التغاير
# 2. Calculate covariance matrix
n_samples = data.shape[0]
covariance = np.cov(mean_centered, rowvar=False)

# 3. تحليل القيم الذاتية
# 3. Eigenvalue decomposition
eigenvals, eigenvecs = np.linalg.eigh(covariance)

# 4. ترتيب تنازلي
# 4. Sort descending
idx = np.argsort(eigenvals)[::-1]
eigenvals = eigenvals[idx]
eigenvecs = eigenvecs[:, idx]

# 5. اختيار المكونات الأولى
# 5. Select top components
components = eigenvecs[:, :n_components]

# 6. إسقاط البيانات على المكونات
# 6. Project data onto components
transformed = np.dot(mean_centered, components)

return {
    "components": components.tolist(),
    "transformed_data": transformed.tolist(),
    "explained_variance": eigenvals[:n_components].tolist(),
```

```
        "explained_variance_ratio": [
            float(ev / sum(eigenvals)) for ev in eigenvals[:n_components]
        ]
    }

@staticmethod
def interpret_eigenvalues(eigenvalues: List[float]) -> dict:
    """
    تفسير القيم الذاتية في سياق البيانات البيولوجية.
    Interpret eigenvalues in the context of biological data.
    """
    total_variance = sum(eigenvalues)
    cumulative = 0.0
    interpretation = []

    for i, ev in enumerate(eigenvalues):
        ratio = ev / total_variance
        cumulative += ratio
        interpretation.append({
            "component": i + 1,
            "eigenvalue": ev,
            "variance_ratio": ratio,
            "cumulative_ratio": cumulative,
            "interpretation": (
                f"المكون {i+1} يفسر {ratio*100:.1f}% من التباين"
            )
        })

    return {
        "total_variance": total_variance,
        "components": interpretation,
        "effective_dimensions": sum(1 for ev in eigenvalues if ev > 1.0)
```

```
    }
```

43.2 ربط التحليل الطيفي بقاعدة البيانات —– اكتشاف العوامل الخفية

43.2 Connecting Spectral Analysis to the Database — Discovering Hidden Factors.

Python

```
class SpectralAnalysisService:
    """
    خدمة تحليل طيفي للبيانات الجينومية.
    Spectral analysis service for genomic data.
    """

    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path
        self.matrix_builder = NumpyExpressionMatrixBuilder(db_path)

    def analyze_expression_patterns(self, n_components: int = 5) -> dict:
        """
        تحليل أنماط التعبير الخفية باستخدام PCA.
        Analyze hidden expression patterns using PCA.
        """
        # جلب جميع العينات والجينات
```

```
        with sqlite3.connect(self.db_path) as conn:
            sample_ids = [row[0] for row in conn.execute("SELECT sample_id FROM 
samples").fetchall()]
            gene_list = [row[0] for row in conn.execute(
                "SELECT DISTINCT gene_symbol FROM read_counts ORDER BY gene_symbol"
            ).fetchall()]

        if not sample_ids or not gene_list:
            return {"error": "لا توجد بيانات كافية"}

        # بناء مصفوفة التعبير
        matrix = self.matrix_builder.build_matrix(sample_ids, gene_list)

        if matrix.rows == 0:
            return {"error": "مصفوفة فارغة"}

        # تحليل PCA
        pca_result = EigenAnalysis.pca_analysis(matrix, n_components)

        # إضافة معلومات عن العينات
        with sqlite3.connect(self.db_path) as conn:
            placeholders = ','.join(['?'] * len(sample_ids))
            query = f"""
                SELECT sample_id, status, tissue_type
                FROM samples
                WHERE sample_id IN ({placeholders})
            """
            rows = conn.execute(query, sample_ids).fetchall()
            sample_metadata = {row[0]: {"status": row[1], "tissue_type": row[2]} for row in rows}

        return {
            "n_samples": len(sample_ids),
```

```
        "n_genes": len(gene_list),
        "pca": pca_result,
        "sample_metadata": sample_metadata,
        "sample_ids": sample_ids,
        "gene_list": gene_list
    }

def detect_batch_effect(self) -> dict:
    """
    الكشف عن تأثيرات الدفعات (Batch Effects) باستخدام تحليل المكونات الرئيسية.
    Detect batch effects using principal component analysis.
    """
    result = self.analyze_expression_patterns(n_components=3)

    if "error" in result:
        return result

    # في الواقع، نفحص ارتباط المكونات الأولى بمتغيرات الدفعة
    # هذا محاكاة تعليمية
    return {
        "batch_effect_detected": False,
        "evidence": "في التحليل الحقيقي، نفحص ارتباط PC1/PC2 بمتغيرات الدفعة",
        "recommendation": "إذا كان التأثير موجوداً، استخدم ComBat أو ضمن الدفعة كنموذج"
    }
```

الفصل الرابع والأربعون: مفهوم التنسور (**Tensor Concept**) — من المصفوفة إلى الأبعاد الأعلى

Chapter Forty-Four: Tensor Concept — From Matrix to Higher Dimensions

الشكل 45 — الخريطة المفاهيمية للفصل

Figure 45 — Conceptual map of the chapter

الانتقال إلى هذا المفهوم يبدأ من مهارة سابقة لديك، استخدمت الإحصاء والمصفوفات في تحليل البيانات؛ الآن ستبني فوق ذلك لغة رياضية للحسابات العلمية.

الشكل 45 — سير العمل الهندسي للفصل

Figure 45 — Engineering workflow of the chapter

Your earlier experience with Python and Biopython prepares you directly for this step, you used statistics and matrices to analyze data; now you will build on them as a mathematical language for scientific computation.

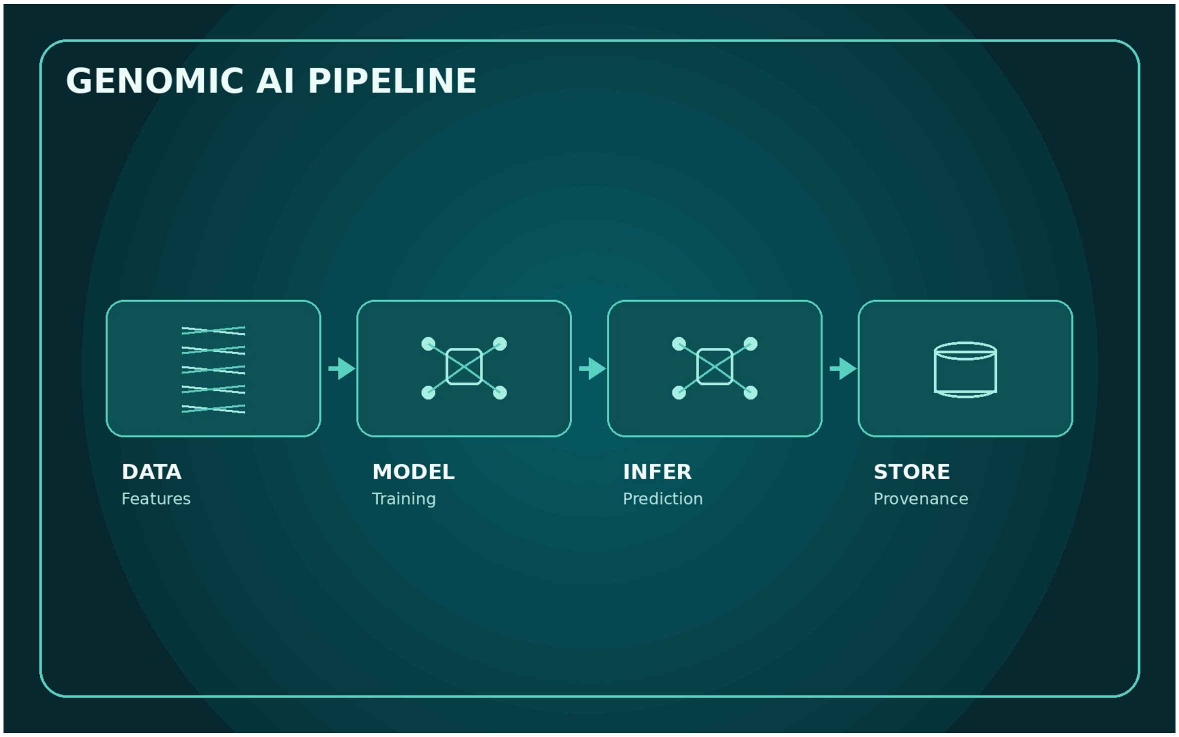


Figure — GENOMIC AI PIPELINE

التنسور (Tensor) هو تعميم للمصفوفة إلى أبعاد أعلى:

· عدد قياسي (Scalar): تنسور من الرتبة 0 (رقم واحد)

· متجه (Vector): تنسور من الرتبة 1 (صف من الأرقام)

· مصفوفة (Matrix): تنسور من الرتبة 2 (جدول من الأرقام)

· تنسور من الرتبة 3: مكعب من الأرقام (مثل مصفوفة عينات × جينات × ظروف)

A Tensor is a generalization of a matrix to higher dimensions:

· Scalar: Rank-0 tensor (a single number).

· Vector: Rank-1 tensor (a row of numbers).

· Matrix: Rank-2 tensor (a table of numbers).

· Rank-3 tensor: A cube of numbers (e.g., samples × genes × conditions).

في المعلوماتية الحيوية، التنسورات تظهر في:

· بيانات متعددة الأوميكس (Multi-omics): عينات × جينات × ميثلية × بروتينات

· بيانات زمنية (Time-series): عينات × جينات × نقاط زمنية

· بيانات مكانية (Spatial transcriptomics): إحداثيات × جينات

In bioinformatics, tensors appear in:

· Multi-omics data: samples × genes × methylation × proteins.

· Time-series data: samples × genes × time points.

· Spatial transcriptomics: coordinates × genes.

لماذا التنسور مهم للجزء الكمومي؟

Why is the tensor important for the quantum part؟

في الحوسبة الكمومية، حالة النظام تُوصف بـ متجه في فضاء هيلبرت (رتبة 1). وعندما يكون لدينا عدة كيوبتات (Qubits)، تصبح الحالة تنسوراً (حاصل ضرب تنسوري للمتجهات الفردية). عمليات البوابات الكمومية هي مصفوفات (رتبة 2) تطبق على هذه المتجهات. وفهم التنسورات هو المفتاح لفك تشابك (Entanglement) وقياس الأنظمة الكمومية.

In quantum computing, the state of a system is described by a vector in Hilbert space (rank 1). When we have multiple Qubits, the state becomes a Tensor (tensor product of individual vectors). Quantum gate operations are Matrices (rank 2) applied to these vectors. Understanding tensors is the key to understanding Entanglement and measuring quantum systems.

44.1 تمثيل التنسور في Python (numpy)

44.1 Tensor Representation in Python (numpy).

Python

```
class TensorRepresentation:
    """تمثيل التنسورات وعملياتها الأساسية.
    Tensor representation and basic operations.
    """

    @staticmethod
    def create_tensor(data: Union[List, np.ndarray]) -> np.ndarray:
        """إنشاء تنسور من بيانات.
        Create a tensor from data.
        """
        if isinstance(data, list):
            return np.array(data, dtype=np.float64)
        return data.astype(np.float64)

    @staticmethod
    def tensor_rank(tensor: np.ndarray) -> int:
        """حساب رتبة التنسور (عدد الأبعاد).
        Calculate tensor rank (number of dimensions).
        """
        return len(tensor.shape)
```

```
@staticmethod
def tensor_shape(tensor: np.ndarray) -> tuple:
    """إرجاع شكل التنسور.
    Return tensor shape.
    """
    return tensor.shape

@staticmethod
def tensor_product(tensor1: np.ndarray, tensor2: np.ndarray) -> np.ndarray:
    """
    حاصل الضرب التنسوري (Kronecker product) لتنسورين.
    Tensor product (Kronecker product) of two tensors.
    """
    return np.kron(tensor1, tensor2)

@staticmethod
def tensor_contraction(tensor: np.ndarray, axes: tuple) -> np.ndarray:
    """
    انكماش تنسوري (جمع على أبعاد محددة) - شبيه بالضرب النقطي المعمم.
    Tensor contraction (summing over specified axes) - similar to generalized dot product.
    """
    return np.tensordot(tensor, tensor, axes=axes)

@staticmethod
def demonstrate_tensor_concepts():
    """عرض مفاهيم التنسور مع أمثلة بيولوجية.
    Demonstrate tensor concepts with biological examples.
    """
    print("=" * 60)
    print("مفاهيم التنسور في المعلوماتية الحيوية")
    print("=" * 60)
```

```
# رتبة 0: عدد قياسي (مثل متوسط تعبير جين واحد)
scalar = np.array([5.0])
print(f"الشكل رتبة 0 (عدد قياسي): {scalar.shape}، القيمة {scalar[0]}")

# رتبة 1: متجه (مثل تعبير عينة واحدة عبر جينات)
vector = np.array([1200.0, 450.0, 2100.0, 800.0])
print(f"الشكل رتبة 1 (متجه): {vector.shape}")

# رتبة 2: مصفوفة (مثل تعبير عدة عينات عبر جينات)
matrix = np.array([
    [1200.0, 450.0, 2100.0, 800.0],
    [980.0, 520.0, 1800.0, 750.0],
    [100.0, 200.0, 80.0, 50.0]
])
print(f"الشكل رتبة 2 (مصفوفة): {matrix.shape}")

# رتبة 3: مكعب (مثل عينات × جينات × ظروف/أوقات)
cube = np.array([
    # الظرف 1 (مريض)
    [[1200.0, 450.0, 2100.0, 800.0],
     [980.0, 520.0, 1800.0, 750.0]],
    # الظرف 2 (سليم)
    [[100.0, 200.0, 80.0, 50.0],
     [110.0, 180.0, 90.0, 60.0]]
])
print(f"الشكل رتبة 3 (مكعب): {cube.shape}")
print("  البعد 0: الظروف (2)")
print("  البعد 1: العينات (2)")
print("  البعد 2: الجينات (4)")

print("\n🔮 التنسورات هي اللغة الطبيعية لـ:")
print("  - بيانات Multi-omics (جينوم + ترانسكريبتوم + بروتيوم)")
```

```
print("  - (تعبير عبر نقاط زمنية متعددة) بيانات زمنية")
print("  - (حالات متعددة الكيوبتات) الحوسبة الكمومية")
```

44.2 من التنسور إلى الحوسبة الكمومية (التمهيد)

44.2 From Tensor to Quantum Computing (Preparation).

في الحوسبة الكمومية، كل كيوبت (Qubit) هو متجه في فضاء ثنائي الأبعاد. عند جمع كيوبتات متعددة، تصبح الحالة الكلية حاصل ضرب تنسوري (Tensor Product) للحالات الفردية.

In quantum computing, each Qubit is a vector in a two-dimensional space. When combining multiple qubits, the overall state becomes the Tensor Product of the individual states.

Code

```
كيوبت واحد: |ψ⟩ = α|0⟩ + β|1⟩   (متجه رتبة 1)
كيوبتان: |ψ⟩ = |ψ₁⟩ ⊗ |ψ₂      (تنسور رتبة 2)
n كيوبتات: |ψ⟩ = |ψ₁⟩ ⊗ ... ⊗ |ψₙ⟩  (تنسور رتبة n)
```

هذا هو الجسر المباشر إلى الباب التاسع: الحوسبة الكمومية ليست سحراً؛ إنها جبر خطي على تنسورات في فضاءات معقدة، مع قواعد فيزيائية خاصة (التراكب، التشابك، القياس).

This is the direct bridge to Unit Nine: Quantum Computing is not magic; it is Linear Algebra on Tensors in Complex Spaces, with special physical rules (Superposition, Entanglement, Measurement).

Python

```
class QuantumPreparation:
    """تمهيد لمفاهيم الحوسبة الكمومية باستخدام التنسورات.
    Preparation for quantum computing concepts using tensors.
    """

    @staticmethod
    def single_qubit_state(alpha: complex, beta: complex) -> np.ndarray:
        """
        إنشاء حالة كيوبت واحد: |ψ⟩ = α|0⟩ + β|1⟩
        Create a single qubit state: |ψ⟩ = α|0⟩ + β|1⟩
        """
        # التأكد من التطبيع: |α|² + |β|² = 1
        norm = np.sqrt(abs(alpha)**2 + abs(beta)**2)
```

```
        if norm > 0:
            alpha = alpha / norm
            beta = beta / norm
        return np.array([alpha, beta], dtype=np.complex128)

    @staticmethod
    def two_qubit_state(state1: np.ndarray, state2: np.ndarray) -> np.ndarray:
        """
        إنشاء حالة كيوبتين باستخدام حاصل الضرب التنسوري.
        Create a two-qubit state using tensor product.
        """
        return np.kron(state1, state2)

    @staticmethod
    def demonstrate_quantum_state():
        """عرض حالة كمومية بسيطة.
        Demonstrate a simple quantum state.
        """
        print("=" * 60)
        print("تمهيد للحوسبة الكمومية: الحالة الكمومية كتنسور")
        print("=" * 60)

        # كيوبت في حالة التراكب: (|0⟩ + |1⟩)/√2
        alpha = 1 / np.sqrt(2)
        beta = 1 / np.sqrt(2)
        qubit = QuantumPreparation.single_qubit_state(alpha, beta)
        print(f"حالة كيوبت واحد: {qubit}")
        print(f"  |0⟩: {abs(qubit[0])**2:.2f}%")
        print(f"  |1⟩: {abs(qubit[1])**2:.2f}%")

        # كيوبتان في حالة التراكب: (|00⟩ + |11⟩) / √2 (حالة تشابك)
        qubit1 = QuantumPreparation.single_qubit_state(1/np.sqrt(2), 1/np.sqrt(2))
```

```
qubit2 = QuantumPreparation.single_qubit_state(1/np.sqrt(2), 1/np.sqrt(2))
two_qubits = QuantumPreparation.two_qubit_state(qubit1, qubit2)
print(f"\nحالة كيوبتين (محاكاة): {two_qubits}")
print("هذه محاكاة تعليمية، التشابك الحقيقي يتطلب بوابات كمومية خاصة.")

print("\n📌 في الحوسبة الكمومية الحقيقية:")
print("  - α و β أعداد مركبة (Complex Numbers)")
print("  - البعد ينمو أسياً: 2^n لـ n كيوبت")
print("  - التنسورات هي اللغة الطبيعية لوصف هذه الحالات")
print("  - في الباب التاسع Quantum Computing هذا هو الباب الذي سندخل منه إلى")
```

الانتقال إلى الباب التاسع:

Transition to Unit Nine:

الآن أصبحت لديك كل الأدوات الرياضية والحاسوبية اللازمة لفهم الحوسبة الكمومية:

· فهمت المتجهات والمصفوفات (البنية الأساسية)

· فهمت القيم الذاتية والتحليل الطيفي (فهم الأنماط)

· فهمت التنسورات (تعميم الأبعاد)

· فهمت الأخطاء العددية والاستقرار (كيف نثق بالنتائج)

في الباب التاسع، نطبق كل هذا على الحوسبة الكمومية نفسها: الكيوبت، التراكب، التشابك، البوابات الكمومية، والدوائر. لن يكون الكم سحراً غامضاً؛ سيكون تطبيقاً طبيعياً للجبر الخطي على تنسورات في فضاء مركب، مع قواعد فيزيائية محددة.

Now you have all the mathematical and computational tools needed to understand Quantum Computing:

· You understood Vectors and Matrices (the basic structure).
· You understood Eigenvalues and Spectral Analysis (understanding patterns).
· You understood Tensors (generalization of dimensions).
· You understood Numerical Errors and Stability (how to trust results).

In Unit Nine, we apply all of this to Quantum Computing itself: Qubits, Superposition, Entanglement, Quantum Gates, and Circuits. Quantum will not

be mysterious magic; it will be a natural application of Linear Algebra to Tensors in Complex Space, with specific physical rules.

—

الباب التاسع: الذكاء الاصطناعي والبيانات الجينومية — من الجبر الخطي إلى النماذج التنبؤية

Unit Nine: Artificial Intelligence and Genomic Data — From Linear Algebra to Predictive Models

الغرض من هذا الباب:

تحويل البيانات المخزنة في SQL (عبر OOP) إلى ميزات (Features) تُغذي خوارزميات التعلم الآلي. سنتعلم كيف ندمج scikit-learn و PyTorch مع مستودعاتنا، وكيف نخزن النماذج المدربة وتنبؤاتها في قاعدة البيانات، وكيف نبني خط أنابيب تعلم آلي (ML Pipeline) قابل للتكرار والتدقيق. هذا هو الجسر

الحقيقي بين "تحليل البيانات" (الجزء الثاني) و"النمذجة التنبؤية" (الجزء الثالث).

Purpose of this unit:

To transform data stored in SQL (via OOP) into Features that feed Machine Learning algorithms. We will learn how to integrate scikit-learn and PyTorch with our repositories, how to store trained models and their predictions in the database, and how to build a reproducible ML Pipeline. This is the real bridge between "data analysis" (Part II) and "predictive modeling" (Part III).

الفصل الخامس والأربعون: بناء مستودع الميزات (Feature Store) باستخدام SQL و OOP

Chapter Forty-Five: Building a Feature Store Using SQL and OOP

مستودع الميزات العلمي ليس مجرد جدول جديد. يجب أن يحتفظ بعلاقة واضحة بين الميزة ومصدرها، وإصدار طريقة حسابها، والزمن الذي كانت فيه متاحة، حتى لا يحدث تسرب للمعلومات من المستقبل إلى تدريب النموذج. إذا حُسبت ميزة باستخدام بيانات لم تكن متاحة وقت التنبؤ، فقد يبدو النموذج ممتازًا في الاختبار بينما يكون أداؤه مضللًا في الاستخدام الحقيقي. لذلك تصبح provenance وtime-aware splits جزءًا من سلامة Feature Store.

A scientific feature store is not simply another table. It must preserve the relationship between a feature, its source, the version of the computation that produced it, and the time at which it became available, so that information does not leak from the future into model training. If a feature uses data that would not have been available at prediction time, the model may look excellent in testing while being misleading in real use. Provenance and time-aware splits are therefore part of Feature Store integrity.

مختبر: تخزين الميزات بالإصدارات

لا يجب أن تمحو الميزة المشتقة القياس الخام أبدًا. خزّن تعريفها وإصدارها وتشغيل المصدر وقيمتها حتى يمكن إعادة بنائها أو مراجعتها.

A derived feature should never erase its raw measurement. Store its definition, version, source run, and value so the feature can be recreated or challenged.

جدول الميزات واجهة علمية، وليس مكانًا لإلقاء قيم عشوائية.

A feature table is a scientific interface, not a dumping ground for arbitrary values.

تخزين ميزات ذات إصدارات

Versioned feature storage

```sql
CREATE TABLE feature_values (
    sample_id TEXT NOT NULL,
    feature_name TEXT NOT NULL,
    feature_value REAL NOT NULL,
    feature_version TEXT NOT NULL,
    derived_from_run INTEGER NOT NULL,
    PRIMARY KEY (sample_id, feature_name, feature_version)
);
```

SQL

الشكل 46 — الخريطة المفاهيمية للفصل

Figure 46 — Conceptual map of the chapter

الآن نعيد استخدام ما سبق، ولكن في سياق هندسي أوسع، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

الشكل 46 — سير العمل الهندسي للفصل

Figure 46 — Engineering workflow of the chapter

This concept does not begin from zero; it begins with a skill you already have, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

قبل أن ندرب أي نموذج، نحتاج إلى ميزات (Features). في علم الأحياء، الميزات هي قيم التعبير الجيني، والطفرات، والبيانات الوصفية (العمر، الجنس، التشخيص). في هذا الفصل، نبني نظاماً يجمع هذه الميزات من جداول متعددة ويحولها إلى مصفوفة جاهزة للتعلم الآلي، مع تخزين تعريف الميزات (Feature Definitions) في SQL لضمان إعادة الإنتاج.

Before training any model, we need Features. In biology, features are gene expression values, mutations, and metadata (age, sex, diagnosis). In this chapter, we build a system that collects these features from multiple tables and transforms them into a matrix ready for machine learning, storing Feature Definitions in SQL to ensure reproducibility.

45.1 هيكل قاعدة بيانات الميزات (Feature Store Schema)

45.1 Feature Store Database Schema.

SQL

```sql
-- ============================================================
-- جدول تعريف الميزات (Feature Definitions)
-- يخزن وصف كل ميزة ومصدرها
-- ============================================================
CREATE TABLE feature_definitions (
    feature_id INTEGER PRIMARY KEY AUTOINCREMENT,
    feature_name TEXT UNIQUE NOT NULL,        -- مثل 'BRCA1_expression', 'age'
    feature_type TEXT CHECK(feature_type IN ('numeric', 'categorical', 'binary')),
    source_table TEXT NOT NULL,               -- 'read_counts', 'patients', 'mutations'
    source_column TEXT NOT NULL,
    extraction_query TEXT,                    -- استعلام SQL لاستخراج الميزة
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);
```

```sql
-- ============================================================
-- جدول تجميع الميزات (Feature Sets)
-- مجموعة من الميزات المستخدمة في نموذج معين
-- ============================================================
CREATE TABLE feature_sets (
    feature_set_id INTEGER PRIMARY KEY AUTOINCREMENT,
    feature_set_name TEXT UNIQUE NOT NULL,
    description TEXT,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);

-- ============================================================
-- جدول الربط بين Feature Sets و Feature Definitions (M:N)
-- ============================================================
CREATE TABLE feature_set_members (
    feature_set_id INTEGER NOT NULL,
    feature_id INTEGER NOT NULL,
    PRIMARY KEY (feature_set_id, feature_id),
    FOREIGN KEY (feature_set_id) REFERENCES feature_sets(feature_set_id) ON DELETE CASCADE,
    FOREIGN KEY (feature_id) REFERENCES feature_definitions(feature_id) ON DELETE CASCADE
);

-- ============================================================
-- جدول تخزين الميزات الفعلية للعينات (Feature Vectors)
-- يمثل المصفوفة النهائية (عينات × ميزات)
-- ============================================================
CREATE TABLE feature_vectors (
    vector_id INTEGER PRIMARY KEY AUTOINCREMENT,
    sample_id INTEGER NOT NULL,
    feature_set_id INTEGER NOT NULL,
    feature_values TEXT NOT NULL,          -- JSON array of values
```

```
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sample_id) REFERENCES samples(sample_id) ON DELETE CASCADE,
    FOREIGN KEY (feature_set_id) REFERENCES feature_sets(feature_set_id) ON DELETE CASCADE,
    UNIQUE (sample_id, feature_set_id)
);

-- فهارس لتسريع الاستعلامات
CREATE INDEX idx_feature_vectors_sample ON feature_vectors(sample_id);
CREATE INDEX idx_feature_vectors_set ON feature_vectors(feature_set_id);
```

45.2 طبقة OOP للميزات (Feature OOP Layer)

45.2 Feature OOP Layer

Python

```
from dataclasses import dataclass, field
from typing import List, Optional, Dict, Any
import json
import sqlite3
import numpy as np

@dataclass
class FeatureDefinition:
    """تعريف ميزة واحدة.
    Single feature definition.
    """
    feature_id: int = 0
```

```
    feature_name: str = ""
    feature_type: str = "numeric"  # numeric, categorical, binary
    source_table: str = ""
    source_column: str = ""
    extraction_query: Optional[str] = None
    created_at: Optional[str] = None

@dataclass
class FeatureSet:
    """مجموعة من الميزات المستخدمة في نموذج معين.
    Set of features used in a specific model.
    """
    feature_set_id: int = 0
    feature_set_name: str = ""
    description: Optional[str] = None
    features: List[FeatureDefinition] = field(default_factory=list)
    created_at: Optional[str] = None

@dataclass
class FeatureVector:
    """متجه الميزات لعينة محددة.
    Feature vector for a specific sample.
    """
    vector_id: int = 0
    sample_id: int = 0
    feature_set_id: int = 0
    values: np.ndarray = field(default_factory=lambda: np.array([]))
    created_at: Optional[str] = None

    def to_json(self) -> str:
        """تحويل القيم إلى JSON للتخزين في SQL.
        Convert values to JSON for SQL storage.
```

```
        """
        return json.dumps(self.values.tolist())

    @classmethod
    def from_json(cls, json_str: str) -> 'FeatureVector':
        """إنشاء FeatureVector من JSON.
        Create FeatureVector from JSON.
        """
        data = json.loads(json_str)
        return cls(values=np.array(data))
```

45.3 مستودع الميزات (Feature Repository)

45.3 Feature Repository

Python

```
class FeatureRepository:
    """مستودع الميزات - يربط SQL بـ OOP ويبني مصفوفات الميزات.
    Feature Repository - bridges SQL to OOP and builds feature matrices.
    """

    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path
        self._ensure_schema()

    def _ensure_schema(self) -> None:
        """التأكد من وجود جداول الميزات (تنشأ تلقائياً).
```

```
        Ensure feature tables exist (created automatically).
        """
        with sqlite3.connect(self.db_path) as conn:
            # تم إنشاؤها في القسم السابق، نضعها هنا للاكتمال
            # Created in previous section, included here for completeness
            conn.executescript("""
                CREATE TABLE IF NOT EXISTS feature_definitions (
                    feature_id INTEGER PRIMARY KEY AUTOINCREMENT,
                    feature_name TEXT UNIQUE NOT NULL,
                    feature_type TEXT CHECK(feature_type IN ('numeric', 'categorical', 'binary')),
                    source_table TEXT NOT NULL,
                    source_column TEXT NOT NULL,
                    extraction_query TEXT,
                    created_at TEXT DEFAULT CURRENT_TIMESTAMP
                );
                CREATE TABLE IF NOT EXISTS feature_sets (
                    feature_set_id INTEGER PRIMARY KEY AUTOINCREMENT,
                    feature_set_name TEXT UNIQUE NOT NULL,
                    description TEXT,
                    created_at TEXT DEFAULT CURRENT_TIMESTAMP
                );
                CREATE TABLE IF NOT EXISTS feature_set_members (
                    feature_set_id INTEGER NOT NULL,
                    feature_id INTEGER NOT NULL,
                    PRIMARY KEY (feature_set_id, feature_id),
                    FOREIGN KEY (feature_set_id) REFERENCES feature_sets(feature_set_id) ON DELETE
CASCADE,
                    FOREIGN KEY (feature_id) REFERENCES feature_definitions(feature_id) ON DELETE
CASCADE
                );
                CREATE TABLE IF NOT EXISTS feature_vectors (
                    vector_id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
                sample_id INTEGER NOT NULL,
                feature_set_id INTEGER NOT NULL,
                feature_values TEXT NOT NULL,
                created_at TEXT DEFAULT CURRENT_TIMESTAMP,
                FOREIGN KEY (sample_id) REFERENCES samples(sample_id) ON DELETE CASCADE,
                FOREIGN KEY (feature_set_id) REFERENCES feature_sets(feature_set_id) ON DELETE
CASCADE,
                UNIQUE (sample_id, feature_set_id)
            );
            CREATE INDEX IF NOT EXISTS idx_feature_vectors_sample ON
feature_vectors(sample_id);
            CREATE INDEX IF NOT EXISTS idx_feature_vectors_set ON
feature_vectors(feature_set_id);
        """)
        conn.commit()

    def define_feature(self, definition: FeatureDefinition) -> int:
        """تسجيل تعريف ميزة جديدة في قاعدة البيانات.
        Register a new feature definition in the database.
        """
        with sqlite3.connect(self.db_path) as conn:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO feature_definitions
                (feature_name, feature_type, source_table, source_column, extraction_query)
                VALUES (?, ?, ?, ?, ?)
            """, (
                definition.feature_name,
                definition.feature_type,
                definition.source_table,
                definition.source_column,
                definition.extraction_query
```

```
        ))
        conn.commit()
        return cursor.lastrowid

def create_feature_set(self, name: str, description: str, feature_names: List[str]) -> int:
    """إنشاء مجموعة ميزات جديدة وربطها بالميزات المحددة.
    Create a new feature set and link it to the specified features.
    """
    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.cursor()

        # 1. إنشاء مجموعة الميزات
        cursor.execute("""
            INSERT INTO feature_sets (feature_set_name, description)
            VALUES (?, ?)
        """, (name, description))
        feature_set_id = cursor.lastrowid

        # 2. جلب معرفات الميزات حسب الأسماء
        placeholders = ','.join(['?'] * len(feature_names))
        cursor.execute(f"""
            SELECT feature_id, feature_name FROM feature_definitions
            WHERE feature_name IN ({placeholders})
        """, feature_names)
        feature_rows = cursor.fetchall()
        feature_map = {row[1]: row[0] for row in feature_rows}

        # 3. ربط الميزات بمجموعة الميزات
        for fname in feature_names:
            if fname in feature_map:
                cursor.execute("""
                    INSERT INTO feature_set_members (feature_set_id, feature_id)
```

```
                VALUES (?, ?)
            """, (feature_set_id, feature_map[fname]))

        conn.commit()
        return feature_set_id

def build_feature_matrix(self, feature_set_id: int, sample_ids: List[int]) -> np.ndarray:
    """
    بناء مصفوفة ميزات (عينات × ميزات) لمجموعة ميزات محددة.
    Build a feature matrix (samples × features) for a specific feature set.
    """
    # 1. جلب تعريفات الميزات لمجموعة الميزات
    with sqlite3.connect(self.db_path) as conn:
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        cursor.execute("""
            SELECT fd.feature_name, fd.feature_type, fd.source_table, fd.source_column
            FROM feature_definitions fd
            JOIN feature_set_members fsm ON fd.feature_id = fsm.feature_id
            WHERE fsm.feature_set_id = ?
            ORDER BY fd.feature_id
        """, (feature_set_id,))
        feature_defs = cursor.fetchall()

    if not feature_defs:
        return np.array([])

    feature_names = [row["feature_name"] for row in feature_defs]
    n_features = len(feature_names)

    # 2. بناء المصفوفة (صف لكل عينة، عمود لكل ميزة)
    matrix = []
```

```
    for sid in sample_ids:
        row = []
        for fdef in feature_defs:
            # استخراج القيمة بناءً على المصدر (هنا تنفيذ مبسط)
            # في الواقع، هذا يحتاج إلى استعلامات ديناميكية حسب الجدول
            value = self._extract_feature_value(sid, fdef)
            row.append(value)
        matrix.append(row)

    return np.array(matrix, dtype=np.float64)

def _extract_feature_value(self, sample_id: int, feature_def: dict) -> float:
    """استخراج قيمة ميزة واحدة لعينة محددة (هندسة داخلية).
    Extract a single feature value for a specific sample (internal logic).
    """
    # هذا تطبيق مبسط جداً. في الواقع، يستخدم استعلامات SQL ديناميكية
    # This is a very simplified implementation. In reality, uses dynamic SQL queries.
    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.cursor()

        if feature_def["source_table"] == "patients":
            # مثال: استخراج العمر من جدول المرضى عبر العينة
            cursor.execute("""
                SELECT p.age FROM patients p
                JOIN samples s ON p.patient_id = s.patient_id
                WHERE s.sample_id = ?
            """, (sample_id,))
            row = cursor.fetchone()
            return float(row[0]) if row and row[0] is not None else 0.0

        elif feature_def["source_table"] == "read_counts":
            # مثال: استخراج تعبير جين معين
```

```
        gene_symbol = feature_def["source_column"]
        cursor.execute("""
            SELECT expression_count FROM read_counts
            WHERE sample_id = ? AND gene_symbol = ?
        """, (sample_id, gene_symbol))
        row = cursor.fetchone()
        return float(row[0]) if row and row[0] is not None else 0.0

    else:
        return 0.0
```

الفصل السادس والأربعون: تدريب النماذج وتخزينها في قاعدة البيانات

Chapter Forty-Six: Training Models and Storing Them in the Database

الشكل 47 — الخريطة المفاهيمية للفصل

Figure 47 — Conceptual map of the chapter

المفاهيم التي بُنيت في الجزء الثاني تأخذ هنا وظيفة جديدة، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

الشكل 47 — سير العمل الهندسي للفصل

Figure 47 — Engineering workflow of the chapter

We now reuse an earlier idea, but in a broader engineering context, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

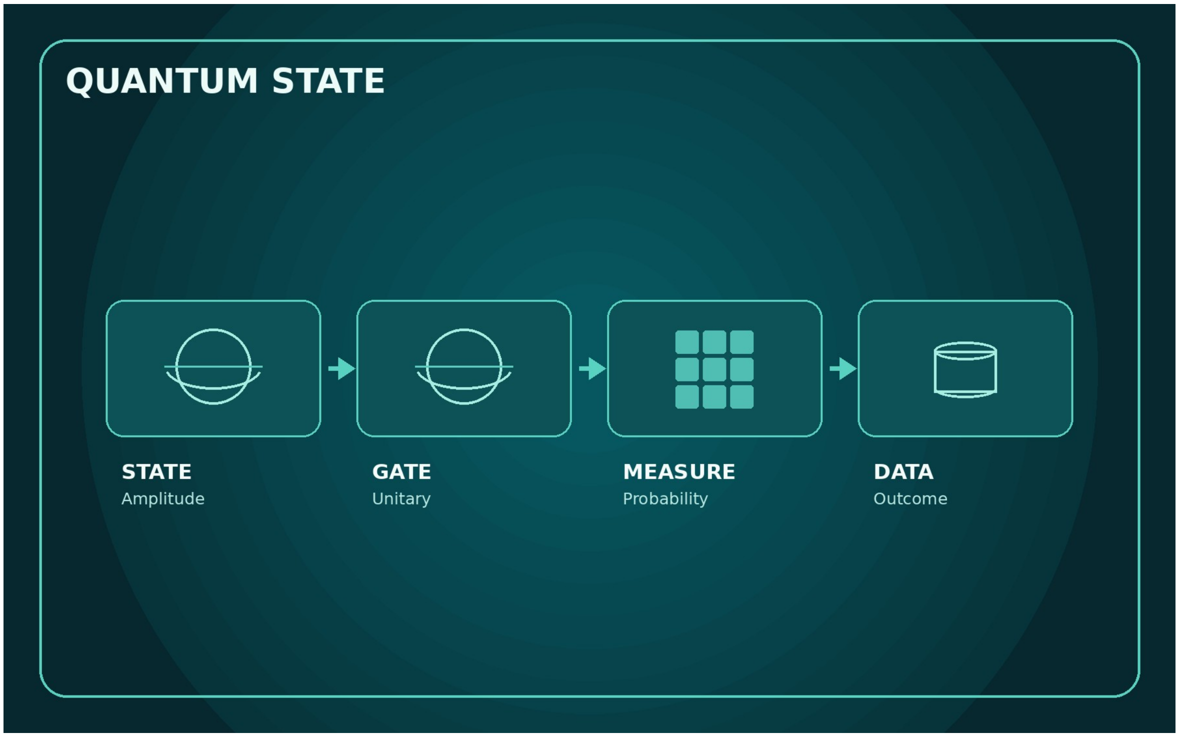


Figure — QUANTUM STATE

46.1 جدول تخزين النماذج (Models Table)

46.1 Models Storage Table

SQL

```
-- ============================================================
-- جدول النماذج المدربة (Trained Models)
-- يخزن النموذج بتنسيق pickle (ثنائي) مع بياناته الوصفية
-- ============================================================
CREATE TABLE trained_models (
   model_id INTEGER PRIMARY KEY AUTOINCREMENT,
   model_name TEXT UNIQUE NOT NULL,
```

```sql
    model_type TEXT NOT NULL,               -- 'sklearn', 'pytorch', 'xgboost'
    feature_set_id INTEGER NOT NULL,
    target_variable TEXT NOT NULL,          -- 'resistance', 'disease_status'
    model_pickle BLOB NOT NULL,             -- النموذج المسلسل (pickle)
    metrics_json TEXT,                      -- {'accuracy': 0.95, 'auc': 0.97}
    hyperparameters_json TEXT,
    training_samples INTEGER,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (feature_set_id) REFERENCES feature_sets(feature_set_id) ON DELETE CASCADE
);

-- فهرس للبحث السريع حسب النموذج ومجموعة الميزات --
CREATE INDEX idx_models_feature_set ON trained_models(feature_set_id);
```

46.2 نموذج تنبؤي كامل مع OOP و SQL (مثال: تصنيف المقاومة)

46.2 Complete Predictive Model with OOP and SQL (Example: Resistance Classification).

Python

```python
import pickle
import json
import sqlite3
from dataclasses import dataclass
```

```
from typing import Optional, Dict, Any, List
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.model_selection import train_test_split

@dataclass
class ModelMetadata:
    """بيانات وصفية للنموذج المدرب.
    Metadata for a trained model.
    """
    model_id: int = 0
    model_name: str = ""
    model_type: str = "sklearn"
    feature_set_id: int = 0
    target_variable: str = ""
    metrics: Dict[str, float] = field(default_factory=dict)
    hyperparameters: Dict[str, Any] = field(default_factory=dict)
    training_samples: int = 0
    created_at: Optional[str] = None

class ModelRepository:
    """مستودع النماذج - يحفظ ويستعيد النماذج المدربة من قاعدة البيانات.
    Model Repository - saves and retrieves trained models from the database.
    """

    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path
        self._ensure_schema()

    def _ensure_schema(self) -> None:
        with sqlite3.connect(self.db_path) as conn:
```

```
        conn.execute("""
            CREATE TABLE IF NOT EXISTS trained_models (
                model_id INTEGER PRIMARY KEY AUTOINCREMENT,
                model_name TEXT UNIQUE NOT NULL,
                model_type TEXT NOT NULL,
                feature_set_id INTEGER NOT NULL,
                target_variable TEXT NOT NULL,
                model_pickle BLOB NOT NULL,
                metrics_json TEXT,
                hyperparameters_json TEXT,
                training_samples INTEGER,
                created_at TEXT DEFAULT CURRENT_TIMESTAMP,
                FOREIGN KEY (feature_set_id) REFERENCES feature_sets(feature_set_id) ON DELETE
CASCADE
            )
        """)
        conn.execute("CREATE INDEX IF NOT EXISTS idx_models_feature_set ON
trained_models(feature_set_id)")
        conn.commit()

    def save_model(self, model: Any, metadata: ModelMetadata) -> int:
        """
        حفظ نموذج مدرب في قاعدة البيانات (مسلسل بـ pickle).
        Save a trained model to the database (serialized with pickle).
        """
        # تسلسل النموذج إلى BLOB
        model_blob = pickle.dumps(model)

        with sqlite3.connect(self.db_path) as conn:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT OR REPLACE INTO trained_models
```

```
            (model_name, model_type, feature_set_id, target_variable,
             model_pickle, metrics_json, hyperparameters_json, training_samples)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?)
        """, (
            metadata.model_name,
            metadata.model_type,
            metadata.feature_set_id,
            metadata.target_variable,
            model_blob,
            json.dumps(metadata.metrics),
            json.dumps(metadata.hyperparameters),
            metadata.training_samples
        ))
        conn.commit()
        return cursor.lastrowid

def load_model(self, model_id: int) -> tuple:
    """
    تحميل نموذج من قاعدة البيانات.
    Load a model from the database.
    """
    with sqlite3.connect(self.db_path) as conn:
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        cursor.execute("""
            SELECT model_pickle, model_name, metrics_json, hyperparameters_json
            FROM trained_models
            WHERE model_id = ?
        """, (model_id,))
        row = cursor.fetchone()

        if not row:
```

```
            raise ValueError(f"Model with ID {model_id} not found")

        model = pickle.loads(row["model_pickle"])
        metadata = ModelMetadata(
            model_id=model_id,
            model_name=row["model_name"],
            metrics=json.loads(row["metrics_json"]),
            hyperparameters=json.loads(row["hyperparameters_json"])
        )
        return model, metadata

def load_latest_model_by_name(self, model_name: str) -> tuple:
    """
    تحميل أحدث إصدار من نموذج بالاسم.
    Load the latest version of a model by name.
    """
    with sqlite3.connect(self.db_path) as conn:
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        cursor.execute("""
            SELECT model_id, model_pickle, model_name, metrics_json, hyperparameters_json
            FROM trained_models
            WHERE model_name = ?
            ORDER BY created_at DESC
            LIMIT 1
        """, (model_name,))
        row = cursor.fetchone()

        if not row:
            raise ValueError(f"Model '{model_name}' not found")

        model = pickle.loads(row["model_pickle"])
```

```
        metadata = ModelMetadata(
            model_id=row["model_id"],
            model_name=row["model_name"],
            metrics=json.loads(row["metrics_json"]),
            hyperparameters=json.loads(row["hyperparameters_json"])
        )
        return model, metadata
```

46.3 خدمة تدريب النماذج (Model Training Service)

46.3 Model Training Service

Python

```
class ModelTrainingService:
    """
    خدمة تدريب النماذج التنبؤية على البيانات الجينومية.
    Predictive model training service for genomic data.
    """

    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path
        self.feature_repo = FeatureRepository(db_path)
        self.model_repo = ModelRepository(db_path)

    def train_resistance_classifier(
        self,
        feature_set_id: int,
```

```
    target_label: str = "resistance",
    test_size: float = 0.2,
    random_state: int = 42
) -> ModelMetadata:
    """
    تدريب نموذج Random Forest لتصنيف المقاومة بناءً على الميزات الجينومية.
    Train a Random Forest classifier for resistance prediction based on genomic features.
    """
    # 1. جلب العينات ذات التصنيف المعروف (مقاومة/غير مقاومة)
    with sqlite3.connect(self.db_path) as conn:
        # نفترض وجود جدول samples مع عمود is_resistant
        # افتراض: جدول resistance_samples يربط sample_id بـ is_resistant
        # هذا هيكل تعليمي؛ في الواقع، قد يكون معقداً أكثر
        cursor = conn.cursor()
        cursor.execute("""
            SELECT s.sample_id,
                   CASE WHEN mr.mutation_id IS NOT NULL THEN 1 ELSE 0 END AS is_resistant
            FROM samples s
            LEFT JOIN sequences seq ON s.sample_id = seq.sample_id
            LEFT JOIN mutations m ON seq.sequence_id = m.sequence_id
            LEFT JOIN mutation_resistance mr ON m.mutation_id = mr.mutation_id
            GROUP BY s.sample_id
        """)
        rows = cursor.fetchall()
        sample_ids = [row[0] for row in rows]
        labels = [row[1] for row in rows]

    if len(sample_ids) < 10:
        raise ValueError("عدد العينات غير كافٍ للتدريب")

    # 2. بناء مصفوفة الميزات
    X = self.feature_repo.build_feature_matrix(feature_set_id, sample_ids)
```

```
y = np.array(labels)

 # 3. تقسيم البيانات
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=test_size, random_state=random_state, stratify=y
)

 # 4. تدريب النموذج
model = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    random_state=random_state,
    n_jobs=-1
)
model.fit(X_train, y_train)

 # 5. تقييم النموذج
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1] if hasattr(model, "predict_proba") else None

metrics = {
    "accuracy": float(accuracy_score(y_test, y_pred)),
    "test_size": float(test_size),
    "training_samples": int(len(X_train))
}

if y_prob is not None and len(np.unique(y_test)) > 1:
    metrics["auc"] = float(roc_auc_score(y_test, y_prob))

 # 6. تخزين النموذج في قاعدة البيانات
metadata = ModelMetadata(
    model_name=f"resistance_classifier_{feature_set_id}_{int(time.time())}",
```

```
        model_type="sklearn",
        feature_set_id=feature_set_id,
        target_variable=target_label,
        metrics=metrics,
        hyperparameters=model.get_params(),
        training_samples=len(X_train)
    )

    self.model_repo.save_model(model, metadata)
    return metadata
```

الفصل السابع والأربعون: التنبؤ (**Inference**) وتخزين النتائج في **SQL**

Chapter Forty-Seven: Inference and Storing Results in SQL

الشكل 48 — الخريطة المفاهيمية للفصل

Figure 48 — Conceptual map of the chapter

من منظور المسار الكامل للكتاب، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

الشكل 48 — سير العمل الهندسي للفصل

Figure 48 — Engineering workflow of the chapter

47.5 الحالة الكمومية والاحتمال — ما الذي يشبه التوزيع الاحتمالي وما الذي لا يشبهه؟

The concepts developed in Part Two take on a new role here, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

47.5 Quantum States and Probability — What Resembles a Probability Distribution and What Does Not?

A qubit can be represented as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are quantum amplitudes and may be complex, with $|\alpha|^2 + |\beta|^2 = 1$. Measurement probabilities are the squared magnitudes of these amplitudes, not the amplitudes themselves. Therefore a quantum state is not simply a classical probability distribution; it also contains phase information that is invisible in the probabilities alone.

يمكن تمثيل كيوبت بحالة من الشكل $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$، حيث α وβ سعتان كموميتان وقد تكونان عقديتين، ويجب أن تحقق $|\alpha|^2 + |\beta|^2 = 1$. الاحتمالات التي تراها عند القياس هي مربعات مقادير السعات، وليست السعات نفسها. لذلك لا يصح أن تقول إن الحالة الكمومية مجرد توزيع احتمالي كلاسيكي؛ فهي تحتوي على معلومات طورية لا تظهر في الاحتمالات وحدها.

IBM Quantum explains that computational-basis measurement yields 0 or 1 with probabilities determined by the squared magnitudes of the overlaps with $|0\rangle$ and $|1\rangle$, and that measurement is fundamentally different from quantum gates.

تشرح مواد IBM Quantum أن القياس في الأساس الحسابي يعطي 0 أو 1 باحتمالين يحددهما مربع مقدار الإسقاط على |0⟩ و|1⟩، وأن القياس عملية مختلفة عن البوابات الكمومية.

Python

```
import numpy as np


alpha = 1 / np.sqrt(2)
beta = 1j / np.sqrt(2)


p0 = abs(alpha) ** 2
p1 = abs(beta) ** 2


print("P(0) =", p0)
print("P(1) =", p1)
print("Total =", p0 + p1)
```

بعد تدريب النموذج، نريد تطبيقه على عينات جديدة (أو بيانات حية) وتخزين التنبؤات في قاعدة البيانات لربطها بالمراجعة البشرية والتوثيق. هذا هو قلب النظام الأخلاقي التنبؤي الذي سنبنيه في الباب العاشر.

After training the model, we want to apply it to new samples (or live data) and store the predictions in the database to link them with human review and documentation. This is the heart of the predictive ethical system we will build in Unit Ten.

47.1 جدول التنبؤات (Predictions Table)

47.1 Predictions Table

SQL

```
-- ============================================================
-- جدول التنبؤات (Predictions) - يخزن نتائج النماذج على عينات محددة
-- ============================================================
CREATE TABLE predictions (
    prediction_id INTEGER PRIMARY KEY AUTOINCREMENT,
    sample_id INTEGER NOT NULL,
    model_id INTEGER NOT NULL,
    prediction_value REAL NOT NULL,        -- الاحتمال أو التصنيف
    prediction_label TEXT,                 -- 'resistant', 'susceptible'
    confidence REAL,                       -- درجة الثقة (0-1)
    input_hash TEXT,                       -- بصمة المدخلات للتتبع
    human_review_status TEXT CHECK(human_review_status IN ('pending', 'approved', 'rejected'))
DEFAULT 'pending',
    human_reviewer_id INTEGER,
    human_decision TEXT,
```

```
    human_review_date TEXT,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sample_id) REFERENCES samples(sample_id) ON DELETE CASCADE,
    FOREIGN KEY (model_id) REFERENCES trained_models(model_id) ON DELETE CASCADE
);

CREATE INDEX idx_predictions_sample ON predictions(sample_id);
CREATE INDEX idx_predictions_model ON predictions(model_id);
CREATE INDEX idx_predictions_status ON predictions(human_review_status);
```

47.2 خدمة التنبؤ بالكامل (Inference Service)

47.2 Complete Inference Service.

Python

```python
import hashlib
import time
from typing import List, Dict, Any

@dataclass
class PredictionResult:
    """نتيجة تنبؤ واحدة.
    Single prediction result.
    """
    sample_id: int
    model_id: int
    prediction_value: float
```

```
    prediction_label: str
    confidence: float
    input_hash: str

class InferenceService:
    """خدمة التنبؤ على العينات الجديدة.
    Inference service for new samples.
    """

    def __init__(self, db_path: str = "genomic_data.db"):
        self.db_path = db_path
        self.feature_repo = FeatureRepository(db_path)
        self.model_repo = ModelRepository(db_path)

    def predict_batch(self, sample_ids: List[int], model_name: str) -> List[PredictionResult]:
        """
        تطبيق نموذج على مجموعة من العينات وتخزين النتائج.
        Apply a model to a batch of samples and store the results.
        """
        # 1. تحميل النموذج
        model, metadata = self.model_repo.load_latest_model_by_name(model_name)

        # 2. بناء مصفوفة الميزات لهذه العينات
        X = self.feature_repo.build_feature_matrix(metadata.feature_set_id, sample_ids)

        if X.shape[0] == 0:
            return []

        # 3. التنبؤ
        predictions = model.predict(X)
        probabilities = model.predict_proba(X) if hasattr(model, "predict_proba") else None
```

```
results = []
with sqlite3.connect(self.db_path) as conn:
    cursor = conn.cursor()

    for i, sid in enumerate(sample_ids):
        pred_val = float(predictions[i])
        prob = float(probabilities[i][1]) if probabilities is not None else 0.5
        label = "resistant" if pred_val == 1 else "susceptible"

        # حساب بصمة المدخلات
        input_hash = hashlib.sha256(
            str(sid).encode() + str(metadata.feature_set_id).encode()
        ).hexdigest()[:16]

        # تخزين التنبؤ في قاعدة البيانات
        cursor.execute("""
            INSERT INTO predictions
            (sample_id, model_id, prediction_value, prediction_label, confidence, input_hash)
            VALUES (?, ?, ?, ?, ?, ?)
        """, (
            sid,
            metadata.model_id,
            pred_val,
            label,
            prob,
            input_hash
        ))
        prediction_id = cursor.lastrowid

        results.append(PredictionResult(
            sample_id=sid,
            model_id=metadata.model_id,
```

```
            prediction_value=pred_val,
            prediction_label=label,
            confidence=prob,
            input_hash=input_hash
        ))

        conn.commit()

    return results

def get_pending_reviews(self) -> List[Dict[str, Any]]:
    """
    استرجاع التنبؤات التي تنتظر المراجعة البشرية.
    Retrieve predictions awaiting human review.
    """
    with sqlite3.connect(self.db_path) as conn:
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        cursor.execute("""
            SELECT p.prediction_id, p.sample_id, p.prediction_label, p.confidence,
                   s.external_id, s.tissue_type,
                   m.model_name,
                   p.created_at
            FROM predictions p
            JOIN samples s ON p.sample_id = s.sample_id
            JOIN trained_models m ON p.model_id = m.model_id
            WHERE p.human_review_status = 'pending'
            ORDER BY p.created_at ASC
            LIMIT 100
        """)
        return [dict(row) for row in cursor.fetchall()]
```

الآن لدينا نظام كامل للتعلم الآلي يعمل مع OOP و SQL. الباب العاشر سيأخذ هذا النظام ويضعه في إطار أخلاقي دفاعي كامل، حيث كل تنبؤ يتم مراجعته، وكل قرار يُسجل، وكل تسلسل يُفحص ضد قواعد السلامة. هذا هو المشروع الواقعي الذي يجمع CRISPR (الجزء الأول)، و biobrick_security (الجزء الثاني)، و OOP+SQL+AI (هذا الباب) في منتج هندسي واحد متكامل.

Now we have a complete machine learning system working with OOP and SQL. Unit Ten will take this system and place it in a complete defensive ethical framework, where every prediction is reviewed, every decision is logged, and every sequence is screened against safety rules. This is the real-world project that brings together CRISPR (Part I), biobrick_security (Part II), and OOP+SQL+AI (this unit) into a single integrated engineering product.

الباب العاشر: المشروع المتكامل — الهندسة الواقعية واسعة النطاق والتوسع قبل الكم

Unit Ten: The Integrated Project — Real-World Large-Scale Engineering and Pre-Quantum Scaling

الغرض من هذا الباب:

بناء المشروع المتكامل النهائي الذي يجمع كل ما تعلمناه في الأجزاء الثلاثة، ولكن هذه المرة على نطاق هندسي حقيقي.

هذا الباب لا يختصر؛ بل يبني نظاماً متكاملاً من طبقات متعددة:

1. طبقة البيانات (SQL + OOP): مخطط متقدم مع جداول مترابطة، وفهارس محسّنة، وإدارة إصدارات (Versioning)، وتخزين JSON للبيانات شبه المهيكلة.

2. طبقة الأعمال (OOP + Services): نماذج مستودعات (Repositories) مع وحدات عمل (Unit of Work) للمعاملات الموزعة، وخدمات تحقق، وتدفقات عمل (Workflows) معقدة.

3. طبقة الحوسبة العلمية (NumPy + SciPy): استخراج البيانات من SQL وتحويلها إلى مصفوفات ضخمة، وإجراء تحليل طيفي (SVD، Eigen-decomposition) على نطاق واسع.

4. طبقة التوسع (البنية التحتية الضخمة): معالجة بالدفعات (Batch Processing)، وفهرسة متقدمة، وتخزين مؤقت (Caching) للنتائج الوسيطة.

5. الجسر إلى الكم (Quantum Bridge): تحديد العمليات الحسابية الأكثر استهلاكاً للوقت (ضرب المصفوفات، تحليل القيم الذاتية) وإثبات رياضي لكيفية تسريعها باستخدام الخوارزميات الكمومية (مثل خوارزمية تقدير الطور، و HHL). هذا الجسر لا يقدم الكم كشيء غامض، بل كحل هندسي لمشكلة أداء محددة قمنا بتشخيصها في نظامنا الكلاسيكي.

Purpose of this unit:

To build the final integrated project that brings together everything learned across all three parts, but this time on a true engineering scale. This unit does not condense; it builds an integrated system across multiple layers:

1. Data Layer (SQL + OOP): An advanced schema with interconnected tables, optimized indexes, versioning, and JSON storage for semi-structured data.
2. Business Layer (OOP + Services): Repository patterns with Unit of Work for distributed transactions, validation services, and complex workflows.

3. Scientific Computing Layer (NumPy + SciPy): Extracting data from SQL and transforming it into massive matrices, performing spectral analysis (SVD, Eigen-decomposition) at scale.

4. Scaling Layer (Massive Infrastructure): Batch Processing, advanced indexing, and caching of intermediate results.

5. The Quantum Bridge: Identifying the most computationally expensive operations (matrix multiplication, eigenvalue decomposition) and providing a mathematical proof of how they can be accelerated using quantum algorithms (e.g., Phase Estimation, HHL). This bridge presents quantum not as magic, but as an engineering solution to a specific performance problem we diagnose in our classical system.

```
from dataclasses import dataclass


@dataclass
class QuantumExperiment:
    problem_name: str
    n_qubits: int
    objective: str
    measurement_basis: str
    repetitions: int


experiment = QuantumExperiment(
    problem_name="toy_energy_estimation",
```

```
    n_qubits=2,
    objective="estimate_ground_state_energy",
    measurement_basis="Z",
    repetitions=1024,
)

print(experiment)
```

Python

Before writing a quantum circuit, define the variables, state space, objective function, measurement target, and validation strategy. These steps prevent quantum computing from becoming a programming experiment without a scientific question.

الشكل 49 — الخريطة المفاهيمية للفصل

Figure 49 — Conceptual map of the chapter

قبل كتابة دائرة كمومية، حدّد المتغيرات، فضاء الحالة، دالة الهدف، ما الذي ستقيسه، وكيف ستتحقق من النتيجة. هذه الخطوات تمنع تحويل quantum computing إلى تجربة برمجية بلا سؤال علمي.

الشكل 49 — سير العمل الهندسي للفصل

Figure 49 — Engineering workflow of the chapter

54.5 من طاقة الجزيء إلى الهاميلتوني —— جسر المفهوم الفيزيائي

54.5 From Molecular Energy to the Hamiltonian — The Physical Bridge

عندما تُدرَس طاقة نظام جزيئي بصورة كلاسيكية، يمكن التعبير عن الاستقرار النسبي من خلال دوال طاقة أو فروق طاقة أو نماذج ترموديناميكية. الانتقال إلى الحوسبة الكمومية لا يعني تحويل هذه القيم مباشرة إلى أرقام داخل دائرة؛ بل يعني أولًا تحديد فضاء الحالات، ثم اختيار تمثيل رياضي للطاقة، ثم بناء مؤثر هاملتوني تكون حالاته الذاتية مرتبطة بمستويات الطاقة التي نريد دراستها. هذه الخطوة تمنع الخلط بين «قيمة طاقة» و«مؤثر طاقة».

When a molecular system is studied classically, relative stability can be described through energy functions, energy differences, or thermodynamic models. Moving to quantum computing does not mean inserting those values directly into a circuit. The workflow first defines a state space, chooses a

mathematical representation of energy, and then constructs a Hamiltonian operator whose eigenstates are associated with the energy levels of interest. This distinction prevents a common category error: confusing an energy value with an energy operator.

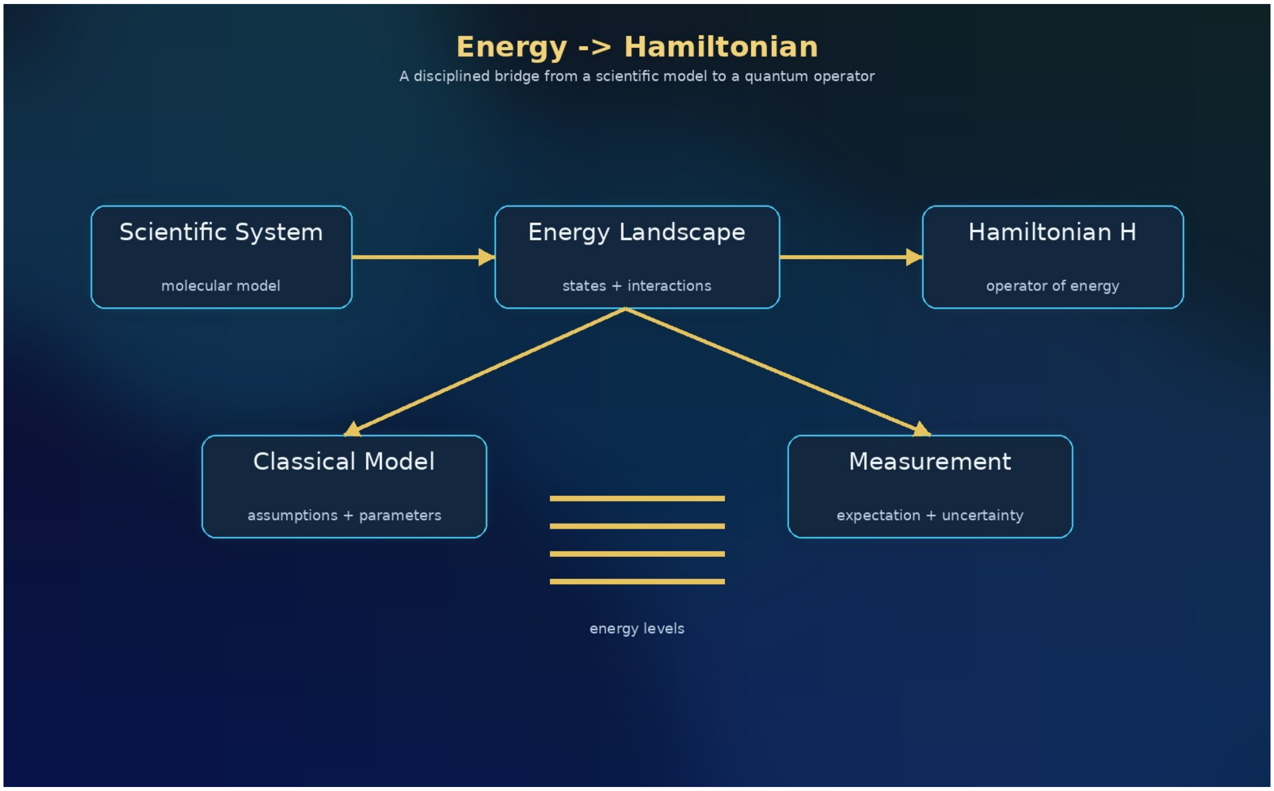
PYTHON

```python
import numpy as np


# A two-level educational Hamiltonian.
# The eigenvalues represent the model's energy levels.
H = np.array([
    [0.0, 0.12],
    [0.12, 0.40]
])


energies, states = np.linalg.eigh(H)


print("Energy levels:", energies)
print("Eigenstates:")
print(states)
```



A scientific energy model prepares the conceptual bridge to the Hamiltonian as the quantum energy operator.

الطاقة في النموذج العلمي تمهّد لفكرة الهاميلتوني بوصفه مشغّل الطاقة في الصياغة الكمومية.

الشكل 50 — الخريطة المفاهيمية للفصل

Figure 50 — Conceptual map of the chapter

في الجبر الخطي المعتاد، تضرب مصفوفة في متجه. في النموذج الكمومي، تصبح البوابات الكمومية مؤثرات خطية

خاصة تحافظ على معيار الحالة، وتصبح الحالة متجهًا في فضاء مركب. هذا التشابه هو الجسر الرياضي الحقيقي بين البابين.

الشكل 50 — سير العمل الهندسي للفصل

Figure 50 — Engineering workflow of the chapter

In ordinary linear algebra, a matrix acts on a vector. In quantum computing, quantum gates are special linear operators that preserve the norm of the state, while the state is a vector in a complex space. This is the genuine mathematical bridge between the two subjects.

Python

```
import numpy as np

H = (1 / np.sqrt(2)) * np.array([
    [1,  1],
    [1, -1]
], dtype=complex)

zero = np.array([1, 0], dtype=complex)
plus = H @ zero

print(plus)
```

Applying Hadamard to |0⟩ produces a superposition. This does not mean that measurement returns a continuous value; measurement produces classical outcomes with defined probabilities.

ستلاحظ أن تطبيق Hadamard على |0⟩ ينتج حالة تراكب. لا يعني ذلك أن القياس يعطي قيمة مستمرة؛ عند القياس ستعود النتيجة إلى مخرجات كلاسيكية ذات احتمالات محددة.

الفصل الثامن والأربعون: البيانات العددية من SQL إلى المصفوفة

Chapter Forty-Eight: Numerical Data from SQL to the Matrix

الفصل التاسع والأربعون: التقييس والتطبيع قبل الجبر الخطي

الشكل 51 — الخريطة المفاهيمية للفصل

Figure 51 — Conceptual map of the chapter

Chapter Forty-Nine: Scaling and Normalization Before Linear Algebra

الشكل 51 — سير العمل الهندسي للفصل

Figure 51 — Engineering workflow of the chapter

From the perspective of the book's complete progression, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

الفصل الخمسون: التباين، التغاير، وPCA

Chapter Fifty: Variance, Covariance, and PCA

هذه الخطوة لا تبدأ من الصفر؛ بل تعتمد على ما سبق بناؤه، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

بعد أن أصبحت البيانات قابلة للاستعلام، تظهر خطوة جديدة: تحويل نتيجة الاستعلام إلى تمثيل عددي مناسب للحساب. هذه الخطوة ليست مجرد تحويل نوع؛ بل اختيار ترتيب للمحاور، وتعريف للعينات والمتغيرات، وتحديد ما إذا كانت القيم الخام قابلة للمقارنة.

Once the data are queryable, a new step appears: transforming query results into a numerical representation suitable for computation. This is not merely a type conversion; it requires defining axes, samples, variables, and whether the raw values are comparable.

Python

```
import numpy as np
```

```
rows = [
    ("S1", "BRCA1", 12.4),
    ("S1", "TP53", 7.1),
    ("S2", "BRCA1", 10.2),
    ("S2", "TP53", 8.9),
]

samples = ["S1", "S2"]
genes = ["BRCA1", "TP53"]

matrix = np.array([
    [12.4, 7.1],
    [10.2, 8.9],
], dtype=float)

print(matrix.shape)
```

Here the relational table becomes a matrix, but the matrix does not automatically carry row and column identities. Metadata must therefore be preserved in a separate layer or an explicit structure.

هنا يصبح الجدول العلاقي مصدرًا لمصفوفة، لكن المصفوفة لا تحمل تلقائيًا أسماء الصفوف والأعمدة؛ لذلك يجب الاحتفاظ بالـmetadata في طبقة منفصلة أو بنية واضحة.

الشكل 52 — الخريطة المفاهيمية للفصل

Figure 52 — Conceptual map of the chapter

الفصل الحادي والخمسون: الاستقرار العددي والتكييف

الشكل 52 — سير العمل الهندسي للفصل

Figure 52 — Engineering workflow of the chapter

Chapter Fifty-One: Numerical Stability and Conditioning

في الموضع الحالي من الرحلة التعليمية، استخدمت الإحصاء والمصفوفات في تحليل البيانات؛ الآن ستبني فوق ذلك لغة رياضية للحسابات العلمية.

This step does not start from scratch; it builds on what came before, you used statistics and matrices to analyze data; now you will build on them as a mathematical language for scientific computation.

ليست كل مصفوفة جاهزة مباشرة لـPCA أو المسافات أو النمذجة. قد تختلف الوحدات والمديات، وقد يكون أحد المتغيرات ذا مقياس أكبر بكثير من الآخر. لذلك يجب أن تعرف لماذا تقوم بالتقييس، ومتى يكون log transformation

مناسبًا، ومتى يصبح التطبيع جزءًا من المنهج العلمي لا مجرد تحسين شكلي.

Not every matrix is immediately ready for PCA, distances, or modeling. Variables may use different scales, and one variable may dominate numerically. You therefore need to understand why scaling is performed, when a log transformation is appropriate, and when normalization is part of the scientific method rather than a cosmetic operation.

Python

```
import numpy as np

X = np.array([
    [10.0, 1000.0],
    [12.0, 1200.0],
    [ 8.0,  900.0],
])

mean = X.mean(axis=0)
std = X.std(axis=0, ddof=1)

Z = (X - mean) / std

print(Z)
```

الشكل 53 — الخريطة المفاهيمية للفصل

Figure 53 — Conceptual map of the chapter

الفصل الثاني والخمسون: التنسورات وضرب Kronecker

الشكل 53 — سير العمل الهندسي للفصل

Figure 53 — Engineering workflow of the chapter

Chapter Fifty-Two: Tensors and the Kronecker Product

بعد العمل السابق على الكائنات والبيانات، استخدمت الإحصاء والمصفوفات في تحليل البيانات؛ الآن ستبني فوق ذلك لغة رياضية للحسابات العلمية.

At this point in the learning path, you used statistics and matrices to analyze data; now you will build on them as a mathematical language for scientific computation.

قبل القفز إلى القيم الذاتية، افهم لماذا تظهر أصلًا. التباين يصف انتشار متغير، بينما التغاير يصف اتجاه تغير متغيرين معًا. مصفوفة التغاير هي التي تجعل سؤال PCA سؤالًا جبريًا خطيًا: ما الاتجاهات التي تحمل أكبر قدر من التباين؟

Before jumping to eigenvalues, understand why they appear. Variance describes the spread of a variable, while covariance describes how two variables vary together. The covariance matrix turns PCA into a linear-algebra question: which directions carry the largest variance?

Python

```
import numpy as np

X = np.array([
    [2.0, 1.0],
    [3.0, 2.0],
    [4.0, 4.0],
    [5.0, 3.0],
])

cov = np.cov(X, rowvar=False)
values, vectors = np.linalg.eigh(cov)

order = np.argsort(values)[::-1]
values = values[order]
vectors = vectors[:, order]

print("Covariance matrix:")
print(cov)
print("Principal values:")
print(values)
```

الشكل 54 — الخريطة المفاهيمية للفصل

Figure 54 — Conceptual map of the chapter

الفصل الثالث والخمسون: من المصفوفة إلى المؤثر الكمومي

الشكل 54 — سير العمل الهندسي للفصل

Figure 54 — Engineering workflow of the chapter

Chapter Fifty-Three: From Matrices to Quantum Operators

الأساس الذي تم بناؤه في الجزء الثاني يسمح لنا الآن بـ استخدمت الإحصاء والمصفوفات في تحليل البيانات؛ الآن ستبني فوق ذلك لغة رياضية للحسابات العلمية.

After working with objects and data in the previous part, you used statistics and matrices to analyze data; now you will build on them as a mathematical language for scientific computation.

Two programs may produce slightly different results even when they use the same equations. Before calling this a programming error, consider conditioning, accumulated round-off, and algorithm choice. An ill-conditioned matrix can amplify small input changes into larger changes in the solution.

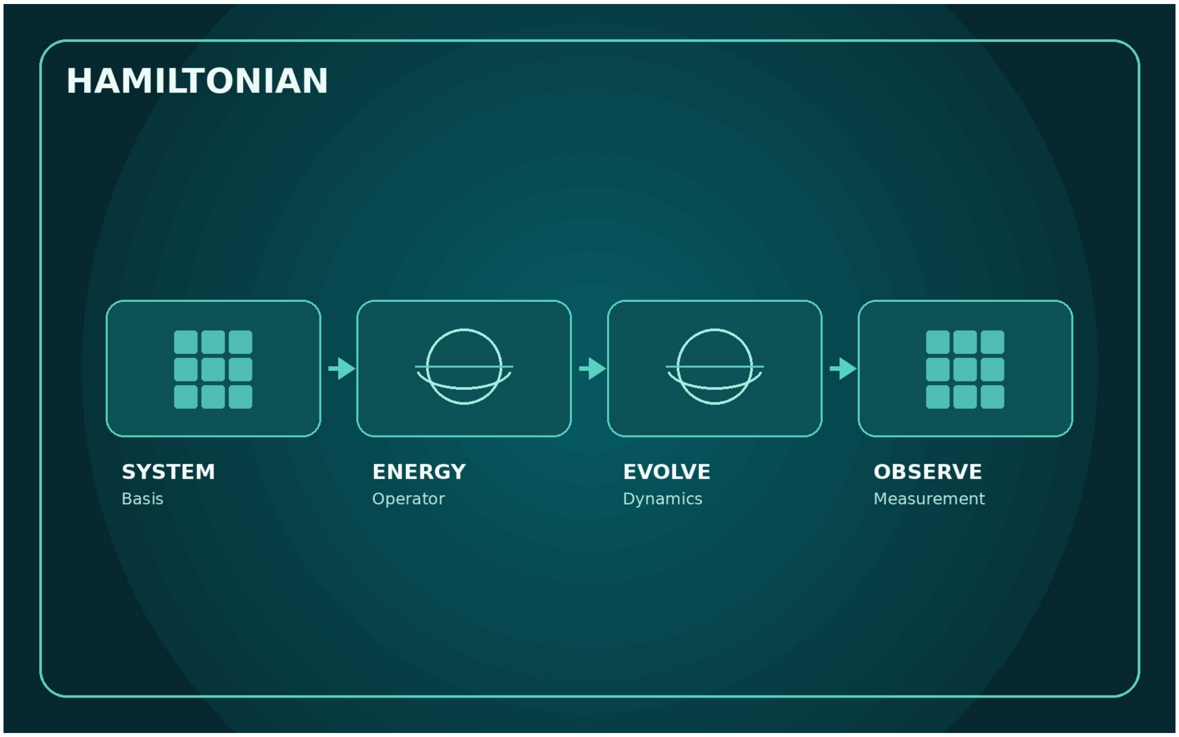


Figure — HAMILTONIAN

قد يعطيك برنامجان نتائج مختلفة قليلًا رغم استخدامهما المعادلات نفسها. قبل أن تعتبر ذلك خطأً برمجيًا، اسأل عن condition number، وتراكم أخطاء التقريب، واختيار الخوارزمية. المصفوفة سيئة التكييف قد تجعل تغيّرًا صغيرًا في المدخلات يؤدي إلى تغير أكبر في الحل.

Python

```
import numpy as np


A = np.array([
```

```
    [1.0, 1.0],
    [1.0, 1.000001]
])

condition_number = np.linalg.cond(A)

print("Condition number:", condition_number)
```

الشكل 55 — الخريطة المفاهيمية للفصل

Figure 55 — Conceptual map of the chapter

الفصل الرابع والخمسون: تهيئة المسألة الكمومية قبل تشغيلها

الشكل 55 — سير العمل الهندسي للفصل

Figure 55 — Engineering workflow of the chapter

Chapter Fifty-Four: Preparing a Quantum Problem Before Execution

هنا يتحول ما تعلمته سابقًا إلى طبقة جديدة من التصميم، تعلمت أن الحساب يجب أن يقود إلى تفسير قابل للتحقق؛ الآن ستطبق المبدأ نفسه عند الانتقال إلى النماذج الكمومية.

The foundation built in Part Two now allows you to you learned that computation must lead to a verifiable interpretation; now you will apply the same principle when entering quantum models.

عندما يصبح النظام متعدد الأبعاد، لا تكفي لغة الصف والعمود وحدها. التنسور تعميم لفكرة المصفوفة، بينما ضرب Kronecker مهم في بناء فضاءات الأنظمة المركبة، وهو مفهوم

ستحتاج إليه لفهم حالات عدة كيوبتات وتمثيل المؤثرات عليها.

When systems become multidimensional, rows and columns are no longer enough. A tensor generalizes the matrix idea, while the Kronecker product is important for constructing composite state spaces and operators, concepts needed for multi-qubit representations.

Python

```
import numpy as np


zero = np.array([1.0, 0.0])
one = np.array([0.0, 1.0])


two_qubit_state = np.kron(zero, one)


print(two_qubit_state)
```

الفصل الخامس والخمسون: التصميم المعماري للمشروع المتكامل — من الصفوف إلى التنسورات

Chapter Fifty-Five: The Integrated Project Architectural Design — From Rows to Tensors

الشكل 56 — الخريطة المفاهيمية للفصل

Figure 56 — Conceptual map of the chapter

الخبرة السابقة مع الملفات والكائنات تصبح في هذا الفصل مدخلًا إلى، استخدمت الإحصاء والمصفوفات في تحليل البيانات؛ الآن ستبني فوق ذلك لغة رياضية للحسابات العلمية.

الشكل 56 — سير العمل الهندسي للفصل

Figure 56 — Engineering workflow of the chapter

Here, an earlier skill becomes a new layer of system design, you used statistics and matrices to analyze data; now you will build on them as a mathematical language for scientific computation.

لن نبدأ بكتابة كود عشوائي. سنبدأ بـ هندسة النظام (System Architecture). المشروع المتكامل يتكون من أربع طبقات أفقية (Layers) وثلاثة ممرات عمودية (Cross-cutting concerns). هذا التصميم يضمن أن كل جزء يمكن اختباره، وتوسيعه، واستبداله (مثل استبدال المحرك الكلاسيكي بمحرك كمومي في المستقبل).

We will not start by writing random code. We will start with System Architecture. The Integrated Project consists of four horizontal Layers and three vertical Cross-cutting concerns. This design ensures that every part can be tested, extended, and replaced (such as replacing the classical engine with a quantum engine in the future).

الطبقات الأفقية (Horizontal Layers):

طبقة العرض (Presentation) CLI، واجهات برمجة التطبيقات (API)، التقارير FastAPI، argparse (مستقبلاً)

طبقة الأعمال (Business) الخدمات، تنسيق العمليات، سير العمل خدمات SecurityOrchestrator، OOP

الطبقة (Layer) المسؤولية (Responsibility) التقنيات (Technologies)

طبقة المستودعات (Repository) الوصول إلى البيانات، التجسير بين OOP و UnitOfWork، SQL Repository

طبقة الحوسبة العدديّة (Numerical) الجبر الخطي، تحليل البيانات، التنسورات Dask، SciPy، NumPy

الممرات العمودية (Cross-cutting Concerns):

الممر (Concern) التوثيق

سجل التدقيق (Audit) كل عملية مسجلة مع بصمة زمنية وهويات

إدارة الإصدارات (Versioning) كل تغيير في البيانات ينتج إصداراً جديداً (نسخة محفوظة)

الأمن والخصوصية (Security) Fail-Closed، تشفير البيانات الحساسة، إخفاء الهوية

55.1 مخطط قاعدة البيانات المتقدمة (نسخة SQL كاملة)

55.1 Advanced Database Schema (Full SQL Version).

سنقوم بإنشاء قاعدة بيانات واحدة موحدة (integrated_project.db) تحتوي على جميع الجداول التي صممناها سابقاً، ولكن مع

إضافة إدارة الإصدارات (Temporal Tables)، و فهارس JSON، و جداول المعاملات. هذا هو العمود الفقري الحقيقي للنظام.

SQL

```
-- ================================================================
-- قاعدة البيانات: integrated_project.db
-- المخطط المتكامل الكامل
-- ================================================================


-- ============================================================
-- 1. الطبقة السريرية (Clinical Layer) - مع إدارة الإصدارات
-- ============================================================


-- جدول المرضى مع تتبع التغييرات (نسخة محفوظة)
CREATE TABLE patients (
    patient_id INTEGER PRIMARY KEY AUTOINCREMENT,
    external_id TEXT UNIQUE NOT NULL,
    age INTEGER CHECK(age >= 0 AND age < 150),
    sex TEXT CHECK(sex IN ('M', 'F', 'O')),
    diagnosis TEXT,
    -- إدارة الإصدارات
    version INTEGER DEFAULT 1,
    valid_from TEXT DEFAULT CURRENT_TIMESTAMP,
    valid_until TEXT,  -- NULL = النسخة الحالية
    is_deleted BOOLEAN DEFAULT 0,
    created_by TEXT
);
```

```sql
-- فهارس مركبة لتسريع الاستعلامات السريرية
CREATE INDEX idx_patients_external ON patients(external_id);
CREATE INDEX idx_patients_diagnosis ON patients(diagnosis);
CREATE INDEX idx_patients_valid ON patients(valid_from, valid_until);

-- جدول العينات
CREATE TABLE samples (
    sample_id INTEGER PRIMARY KEY AUTOINCREMENT,
    patient_id INTEGER NOT NULL,
    tissue_type TEXT NOT NULL,
    collection_date TEXT,
    rna_quality REAL CHECK(rna_quality BETWEEN 0 AND 10),
    status TEXT CHECK(status IN ('control', 'diseased', 'treated')),
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (patient_id) REFERENCES patients(patient_id) ON DELETE CASCADE
);
CREATE INDEX idx_samples_patient ON samples(patient_id);
CREATE INDEX idx_samples_status ON samples(status);

-- ============================================================
-- 2. الطبقة الجزيئية (Molecular Layer) - مع JSON للبيانات الوصفية
-- ============================================================

-- جدول الجينات (مرجعي، يتغير نادراً)
CREATE TABLE genes (
    gene_id INTEGER PRIMARY KEY AUTOINCREMENT,
    gene_symbol TEXT UNIQUE NOT NULL,
    chromosome TEXT NOT NULL,
    start_position INTEGER,
    end_position INTEGER,
    strand INTEGER NOT NULL CHECK(strand IN (1, -1)),
    description TEXT,
```

```
    external_id TEXT UNIQUE,
    -- JSON للبيانات الوصفية المتغيرة (مثل المسارات الحيوية)
    metadata_json TEXT,  -- {'pathways': ['p53', 'DNA_repair'], 'family': 'tumor_suppressor'}
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);
CREATE INDEX idx_genes_symbol ON genes(gene_symbol);
-- فهرس JSON لتسريع البحث داخل metadata_json (SQLite 3.38+)
-- CREATE INDEX idx_genes_metadata ON genes(json_extract(metadata_json, '$.pathways'));

-- جدول التسلسلات مع إدارة الإصدارات والـ Provenance
CREATE TABLE sequences (
    sequence_id INTEGER PRIMARY KEY AUTOINCREMENT,
    sample_id INTEGER NOT NULL,
    gene_id INTEGER NOT NULL,
    run_id INTEGER,  -- من طبقة التجارب
    sequence_text TEXT NOT NULL,
    length INTEGER,
    gc_content REAL CHECK(gc_content BETWEEN 0 AND 100),
    -- إصدار التسلسل (لمتابعة التغييرات في قاعدة البيانات المرجعية)
    version INTEGER DEFAULT 1,
    valid_from TEXT DEFAULT CURRENT_TIMESTAMP,
    valid_until TEXT,
    -- بصمة لمنع التلاعب
    checksum TEXT,  -- SHA-256 للـ sequence_text
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sample_id) REFERENCES samples(sample_id) ON DELETE CASCADE,
    FOREIGN KEY (gene_id) REFERENCES genes(gene_id) ON DELETE CASCADE
);
CREATE INDEX idx_sequences_sample ON sequences(sample_id);
CREATE INDEX idx_sequences_gene ON sequences(gene_id);
CREATE INDEX idx_sequences_checksum ON sequences(checksum);
```

```sql
-- ============================================================
-- 3. طبقة الطفرات والمقاومة (Mutation & Resistance Layer)
-- ============================================================

CREATE TABLE mutations (
    mutation_id INTEGER PRIMARY KEY AUTOINCREMENT,
    sequence_id INTEGER NOT NULL,
    position INTEGER NOT NULL,
    ref_base TEXT NOT NULL CHECK(ref_base IN ('A', 'T', 'C', 'G', 'U')),
    alt_base TEXT NOT NULL CHECK(alt_base IN ('A', 'T', 'C', 'G', 'U')),
    effect TEXT CHECK(effect IN ('missense', 'nonsense', 'silent', 'frameshift')),
    -- تأثير سريري محتمل (JSON)
    clinical_impact_json TEXT,  -- {'clinvar_id': '123', 'significance': 'pathogenic'}
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sequence_id) REFERENCES sequences(sequence_id) ON DELETE CASCADE,
    UNIQUE (sequence_id, position)
);
CREATE INDEX idx_mutations_sequence ON mutations(sequence_id);
CREATE INDEX idx_mutations_effect ON mutations(effect);

CREATE TABLE resistance (
    resistance_id INTEGER PRIMARY KEY AUTOINCREMENT,
    drug_name TEXT UNIQUE NOT NULL,
    drug_class TEXT,
    description TEXT
);

-- الجدول الوسيط (M:N) مع بيانات إضافية
CREATE TABLE mutation_resistance (
    mutation_id INTEGER NOT NULL,
    resistance_id INTEGER NOT NULL,
    confidence REAL DEFAULT 1.0 CHECK(confidence BETWEEN 0 AND 1),
```

```sql
    evidence TEXT,  -- PMID أو رابط
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (mutation_id, resistance_id),
    FOREIGN KEY (mutation_id) REFERENCES mutations(mutation_id) ON DELETE CASCADE,
    FOREIGN KEY (resistance_id) REFERENCES resistance(resistance_id) ON DELETE CASCADE
);

-- ============================================================
-- 4. الطبقة التجريبية (Experimental Layer) - التتبع الكامل
-- ============================================================

CREATE TABLE experiments (
    experiment_id INTEGER PRIMARY KEY AUTOINCREMENT,
    sample_id INTEGER NOT NULL,
    protocol_name TEXT NOT NULL,
    library_type TEXT CHECK(library_type IN ('stranded', 'unstranded')),
    target_read_length INTEGER,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sample_id) REFERENCES samples(sample_id) ON DELETE CASCADE
);
CREATE INDEX idx_experiments_sample ON experiments(sample_id);

CREATE TABLE runs (
    run_id INTEGER PRIMARY KEY AUTOINCREMENT,
    experiment_id INTEGER NOT NULL,
    instrument_model TEXT NOT NULL,
    run_date TEXT,
    total_reads INTEGER,
    total_bases INTEGER,
    q30_percent REAL CHECK(q30_percent BETWEEN 0 AND 100),
    fastq_path TEXT,
    checksum TEXT,  -- SHA-256 للملف الخام
```

```
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (experiment_id) REFERENCES experiments(experiment_id) ON DELETE
CASCADE
);
CREATE INDEX idx_runs_experiment ON runs(experiment_id);

-- ============================================================
-- 5. الطبقة الأمنية والأخلاقية (Security & Ethical Layer)
-- ============================================================

-- السياسات الأمنية
CREATE TABLE security_policies (
    policy_id INTEGER PRIMARY KEY AUTOINCREMENT,
    policy_name TEXT UNIQUE NOT NULL,
    policy_type TEXT CHECK(policy_type IN ('sequence_pattern', 'ai_threshold',
'similarity_threshold')),
    rule_definition TEXT NOT NULL,  -- JSON
    risk_level TEXT CHECK(risk_level IN ('low', 'medium', 'high')) DEFAULT 'medium',
    is_active BOOLEAN DEFAULT 1,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    created_by TEXT
);

-- قائمة انتظار المراجعة البشرية
CREATE TABLE review_queue (
    review_id INTEGER PRIMARY KEY AUTOINCREMENT,
    sequence_id INTEGER NOT NULL,
    sample_id INTEGER NOT NULL,
    policy_id INTEGER,
    risk_level TEXT NOT NULL,
    -- نتائج الفحص مخزنة كـ JSON للشفافية
    screening_results_json TEXT,  -- {'pattern': {...}, 'ai': {...}, 'similarity': {...}}
```

```
    input_hash TEXT NOT NULL,
    status TEXT CHECK(status IN ('pending', 'in_progress', 'approved', 'rejected', 'more_info'))
DEFAULT 'pending',
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sequence_id) REFERENCES sequences(sequence_id) ON DELETE CASCADE,
    FOREIGN KEY (sample_id) REFERENCES samples(sample_id) ON DELETE CASCADE,
    FOREIGN KEY (policy_id) REFERENCES security_policies(policy_id) ON DELETE SET NULL
);
CREATE INDEX idx_review_queue_status ON review_queue(status);
CREATE INDEX idx_review_queue_risk ON review_queue(risk_level);

-- المراجعات البشرية (واحد لواحد مع review_queue)
CREATE TABLE human_reviews (
    review_id INTEGER PRIMARY KEY AUTOINCREMENT,
    queue_id INTEGER UNIQUE NOT NULL,
    reviewer_id INTEGER NOT NULL,
    decision TEXT CHECK(decision IN ('approved', 'rejected', 'more_info')) NOT NULL,
    rationale TEXT NOT NULL,
    reviewed_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (queue_id) REFERENCES review_queue(review_id) ON DELETE CASCADE,
    FOREIGN KEY (reviewer_id) REFERENCES users(user_id) ON DELETE CASCADE
);

-- ============================================================
-- 6. الطبقة الإدارية (Administrative Layer) - التدقيق المتقدم
-- ============================================================

-- جدول المستخدمين
CREATE TABLE users (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE NOT NULL,
    role TEXT CHECK(role IN ('admin', 'researcher', 'reviewer', 'auditor')) DEFAULT 'researcher',
```

```sql
    public_key TEXT,  -- للتوقيع الرقمي (محاكاة)
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);

-- سجل التدقيق المتقدم (مع بصمات لا رجعة فيها) --
CREATE TABLE audit_log (
    audit_id INTEGER PRIMARY KEY AUTOINCREMENT,
    run_id TEXT NOT NULL,
    actor TEXT NOT NULL,
    action TEXT NOT NULL,
    target_table TEXT,
    target_id TEXT,
    old_value_hash TEXT,  -- SHA-256 للقيمة القديمة
    new_value_hash TEXT,  -- SHA-256 للقيمة الجديدة
    rationale TEXT,
    software_version TEXT,
    timestamp TEXT DEFAULT CURRENT_TIMESTAMP,
    -- بصمة للسجل نفسه (لمنع التلاعب) --
    previous_audit_hash TEXT,
    current_audit_hash TEXT
);
CREATE INDEX idx_audit_actor ON audit_log(actor);
CREATE INDEX idx_audit_run ON audit_log(run_id);
CREATE INDEX idx_audit_time ON audit_log(timestamp);
```

الفصل السادس والخمسون: طبقة **OOP** المتقدمة — المستودعات، ووحدات العمل، والمصانع

Chapter Fifty-Six: Advanced OOP Layer — Repositories, Unit of Work, and Factories

مختبر: وحدة العمل والمصانع

Laboratory: Unit of Work and Factories

عندما تشترك عدة مستودعات في حدث علمي واحد، تنسق Unit of Work المعاملة المشتركة. القيمة هنا هي الذرية، لا التجريد لمجرد التجريد.

When several repositories participate in one scientific event, a Unit of Work coordinates the shared transaction. The value is atomicity, not abstraction for its own sake.

تفيد المصانع عندما تختلف صيغ المصادر بينما يجب أن يظل كائن المجال ثابتًا.

Factories are useful when source formats differ but the domain object should remain stable.

تنسيق عملية مجال ذرية

Coordinating an atomic domain operation

```
class UnitOfWork:
    def __init__(self, connection, repositories):
        self.connection = connection
        self.repositories = repositories

    def __enter__(self):
        self.connection.execute('BEGIN')
        return self

    def __exit__(self, exc_type, exc, tb):
        if exc_type:
            self.connection.rollback()
        else:
            self.connection.commit()
```

PYTHON

الشكل 57 — الخريطة المفاهيمية للفصل

Figure 57 — Conceptual map of the chapter

الآن نستخدم المهارة السابقة للوصول إلى مستوى جديد من النظام. بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

الشكل 57 — سير العمل الهندسي للفصل

Figure 57 — Engineering workflow of the chapter

Your previous experience with files and objects now becomes an entry point to you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

هذا الفصل يبني هيكلاً OOP متكاملاً لا يقتصر على مستودعات بسيطة، بل يتضمن وحدات عمل (Unit of Work) لإدارة المعاملات الموزعة، و مصانع (Factories) لإنشاء الكائنات المعقدة، و مقدمي خدمات (Service Providers) لتنسيق كل شيء.

56.1 كائنات النموذج الأساسية (Core Model Objects)

56.1 Core Model Objects

Python

```
from dataclasses import dataclass, field
from typing import Optional, List, Dict, Any
from enum import Enum
import json
```

```
import hashlib


class RiskLevel(Enum):
    LOW = "low"
    MEDIUM = "medium"
    HIGH = "high"


class ReviewStatus(Enum):
    PENDING = "pending"
    APPROVED = "approved"
    REJECTED = "rejected"


@dataclass
class Patient:
    patient_id: int = 0
    external_id: str = ""
    age: Optional[int] = None
    sex: Optional[str] = None
    diagnosis: Optional[str] = None
    version: int = 1
    valid_from: Optional[str] = None
    valid_until: Optional[str] = None
    is_deleted: bool = False
    created_by: str = "system"

    def to_dict(self) -> Dict[str, Any]:
        return {
            "patient_id": self.patient_id,
            "external_id": self.external_id,
            "age": self.age,
            "sex": self.sex,
            "diagnosis": self.diagnosis,
```

```
            "version": self.version,
            "valid_from": self.valid_from,
            "valid_until": self.valid_until
        }

@dataclass
class Gene:
    gene_id: int = 0
    gene_symbol: str = ""
    chromosome: str = ""
    start_position: Optional[int] = None
    end_position: Optional[int] = None
    strand: int = 1
    description: Optional[str] = None
    external_id: Optional[str] = None
    metadata: Dict[str, Any] = field(default_factory=dict)

    def metadata_to_json(self) -> str:
        return json.dumps(self.metadata)

@dataclass
class Sequence:
    sequence_id: int = 0
    sample_id: int = 0
    gene_id: int = 0
    run_id: Optional[int] = None
    sequence_text: str = ""
    length: int = 0
    gc_content: float = 0.0
    version: int = 1
    valid_from: Optional[str] = None
    valid_until: Optional[str] = None
```

```
    checksum: str = ""

    def __post_init__(self):
        if not self.checksum and self.sequence_text:
            self.checksum = hashlib.sha256(self.sequence_text.encode()).hexdigest()
        if self.length == 0 and self.sequence_text:
            self.length = len(self.sequence_text)

@dataclass
class Mutation:
    mutation_id: int = 0
    sequence_id: int = 0
    position: int = 0
    ref_base: str = ""
    alt_base: str = ""
    effect: Optional[str] = None
    clinical_impact: Dict[str, Any] = field(default_factory=dict)

@dataclass
class ReviewQueueItem:
    review_id: int = 0
    sequence_id: int = 0
    sample_id: int = 0
    policy_id: Optional[int] = None
    risk_level: RiskLevel = RiskLevel.MEDIUM
    screening_results: Dict[str, Any] = field(default_factory=dict)
    input_hash: str = ""
    status: ReviewStatus = ReviewStatus.PENDING
    created_at: Optional[str] = None
```

56.2 المستودع المتقدم مع وحدات العمل (Unit of Work)

56.2 Advanced Repository with Unit of Work.

Python

```
import sqlite3
from typing import List, Optional, Dict, Any
from contextlib import contextmanager

class UnitOfWork:
    """
    وحدة العمل - تدير معاملة قاعدة البيانات وتضمن Atomicity.
    Unit of Work - Manages the database transaction and ensures Atomicity.
    جميع المستودعات تشترك في نفس الاتصال خلال المعاملة.
    All repositories share the same connection during the transaction.
    """
    def __init__(self, db_path: str = "integrated_project.db"):
        self.db_path = db_path
        self.connection = None
        self.repositories = {}

    def __enter__(self):
        self.connection = sqlite3.connect(self.db_path)
        self.connection.row_factory = sqlite3.Row
        # بدء المعاملة
        self.connection.execute("BEGIN TRANSACTION")
        # إنشاء المستودعات مع هذا الاتصال
        self.repositories['patient'] = PatientRepository(self.connection)
        self.repositories['sequence'] = SequenceRepository(self.connection)
```

```
        self.repositories['mutation'] = MutationRepository(self.connection)
        self.repositories['audit'] = AuditRepository(self.connection)
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        if exc_type:
            self.connection.rollback()
        else:
            self.connection.commit()
        self.connection.close()

    def get_repository(self, name: str):
        return self.repositories.get(name)


class PatientRepository:
    """مستودع المرضى مع دعم الإصدارات (Temporal Versioning)."""
    def __init__(self, connection: sqlite3.Connection):
        self.connection = connection

    def save_new_version(self, patient: Patient, actor: str) -> int:
        """حفظ نسخة جديدة من المريض (إغلاق القديم، إنشاء جديد)."""
        cursor = self.connection.cursor()

        # 1. إغلاق الإصدار القديم (إن كان موجوداً)
        if patient.patient_id > 0:
            cursor.execute("""
                UPDATE patients
                SET valid_until = CURRENT_TIMESTAMP
                WHERE patient_id = ? AND valid_until IS NULL
            """, (patient.patient_id,))
```

```
        # 2. إنشاء الإصدار الجديد
        cursor.execute("""
            INSERT INTO patients
            (external_id, age, sex, diagnosis, version, valid_from, created_by)
            VALUES (?, ?, ?, ?, ?, CURRENT_TIMESTAMP, ?)
        """, (
            patient.external_id,
            patient.age,
            patient.sex,
            patient.diagnosis,
            patient.version + 1,
            actor
        ))
        return cursor.lastrowid

    def get_current(self, patient_id: int) -> Optional[Patient]:
        cursor = self.connection.cursor()
        cursor.execute("""
            SELECT * FROM patients
            WHERE patient_id = ? AND valid_until IS NULL AND is_deleted = 0
        """, (patient_id,))
        row = cursor.fetchone()
        if not row:
            return None
        return Patient(
            patient_id=row["patient_id"],
            external_id=row["external_id"],
            age=row["age"],
            sex=row["sex"],
            diagnosis=row["diagnosis"],
            version=row["version"],
            valid_from=row["valid_from"],
```

```
            valid_until=row["valid_until"],
            created_by=row["created_by"]
        )


class SequenceRepository:
    """مستودع التسلسلات مع دعم Checksum والتحقق من الصحة."""
    def __init__(self, connection: sqlite3.Connection):
        self.connection = connection

    def save(self, sequence: Sequence) -> int:
        cursor = self.connection.cursor()
        cursor.execute("""
            INSERT INTO sequences
            (sample_id, gene_id, run_id, sequence_text, length, gc_content,
             version, valid_from, checksum)
            VALUES (?, ?, ?, ?, ?, ?, ?, CURRENT_TIMESTAMP, ?)
        """, (
            sequence.sample_id,
            sequence.gene_id,
            sequence.run_id,
            sequence.sequence_text,
            sequence.length,
            sequence.gc_content,
            sequence.version,
            sequence.checksum
        ))
        return cursor.lastrowid

    def find_by_checksum(self, checksum: str) -> Optional[Sequence]:
        cursor = self.connection.cursor()
        cursor.execute("""
```

```
        SELECT * FROM sequences
        WHERE checksum = ? AND valid_until IS NULL
    """, (checksum,))
    row = cursor.fetchone()
    if not row:
        return None
    return Sequence(
        sequence_id=row["sequence_id"],
        sample_id=row["sample_id"],
        gene_id=row["gene_id"],
        run_id=row["run_id"],
        sequence_text=row["sequence_text"],
        length=row["length"],
        gc_content=row["gc_content"],
        version=row["version"],
        valid_from=row["valid_from"],
        valid_until=row["valid_until"],
        checksum=row["checksum"]
    )

def get_all_sequences_for_analysis(self, limit: int = 1000, offset: int = 0) -> List[Sequence]:
    """استرجاع تسلسلات للتحليل العددي مع الترحيل (Pagination)."""
    cursor = self.connection.cursor()
    cursor.execute("""
        SELECT * FROM sequences
        WHERE valid_until IS NULL
        ORDER BY sequence_id
        LIMIT ? OFFSET ?
    """, (limit, offset))
    rows = cursor.fetchall()
    return [
        Sequence(
```

```
            sequence_id=r["sequence_id"],
            sample_id=r["sample_id"],
            gene_id=r["gene_id"],
            run_id=r["run_id"],
            sequence_text=r["sequence_text"],
            length=r["length"],
            gc_content=r["gc_content"],
            version=r["version"],
            valid_from=r["valid_from"],
            valid_until=r["valid_until"],
            checksum=r["checksum"]
        ) for r in rows
    ]


class AuditRepository:
    """مستودع التدقيق مع سلاسل التجزئة المترابطة (Blockchain-like Audit)."""
    def __init__(self, connection: sqlite3.Connection):
        self.connection = connection
        self._last_hash = self._get_last_hash()

    def _get_last_hash(self) -> str:
        cursor = self.connection.cursor()
        cursor.execute("SELECT current_audit_hash FROM audit_log ORDER BY audit_id DESC LIMIT 
1")
        row = cursor.fetchone()
        return row[0] if row else "GENESIS_HASH"

    def log(self, run_id: str, actor: str, action: str, target_table: str,
            target_id: str, old_value_hash: str = "", new_value_hash: str = "",
            rationale: str = "", software_version: str = "1.0.0") -> None:
        cursor = self.connection.cursor()
```

```
# حساب بصمة السجل الحالي (بما فيها البصمة السابقة)
combined = f"{self._last_hash}|{run_id}|{actor}|{action}|{target_table}|{target_id}"
current_hash = hashlib.sha256(combined.encode()).hexdigest()

cursor.execute("""
    INSERT INTO audit_log
    (run_id, actor, action, target_table, target_id,
     old_value_hash, new_value_hash, rationale, software_version,
     previous_audit_hash, current_audit_hash)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
""", (
    run_id, actor, action, target_table, target_id,
    old_value_hash, new_value_hash, rationale, software_version,
    self._last_hash, current_hash
))
self._last_hash = current_hash
```

الفصل السابع والخمسون: الحوسبة العلمية على نطاق واسع — استخراج المصفوفات من SQL

Chapter Fifty-Seven: Large-Scale Scientific Computing — Extracting Matrices from SQL

يخلق استخراج مصفوفة كبيرة من SQL نقطة اختناق جديدة: قد تكون قاعدة البيانات قادرة على الإجابة بسرعة، بينما تفشل الذاكرة عند محاولة تحميل النتيجة كاملة. الحل الهندسي ليس دائمًا «حاسوبًا أكبر»، بل تصميم تدفق يسمح بالتقسيم إلى دفعات أو streaming، مع تحديد ترتيب الصفوف والأعمدة، والتحقق من عدم فقدان السجلات عند إعادة التجميع. يجب أن يكون التحويل من relational representation إلى matrix representation محددًا وقابلًا لإعادة الإنتاج.

Extracting a large matrix from SQL can create a new bottleneck: the database may answer efficiently while memory fails when the entire result is loaded at once. The engineering answer is not always “use a larger computer”; it may be to stream or batch the extraction while explicitly defining row and column order and checking that no records are lost during reconstruction. The transformation from a relational representation to a matrix representation must be deterministic and reproducible.

الشكل 58 — الخريطة المفاهيمية للفصل

Figure 58 — Conceptual map of the chapter

في هذه المرحلة من المسار، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

الشكل 58 — سير العمل الهندسي للفصل

Figure 58 — Engineering workflow of the chapter

We now use the earlier skill to reach the next level of the system. you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

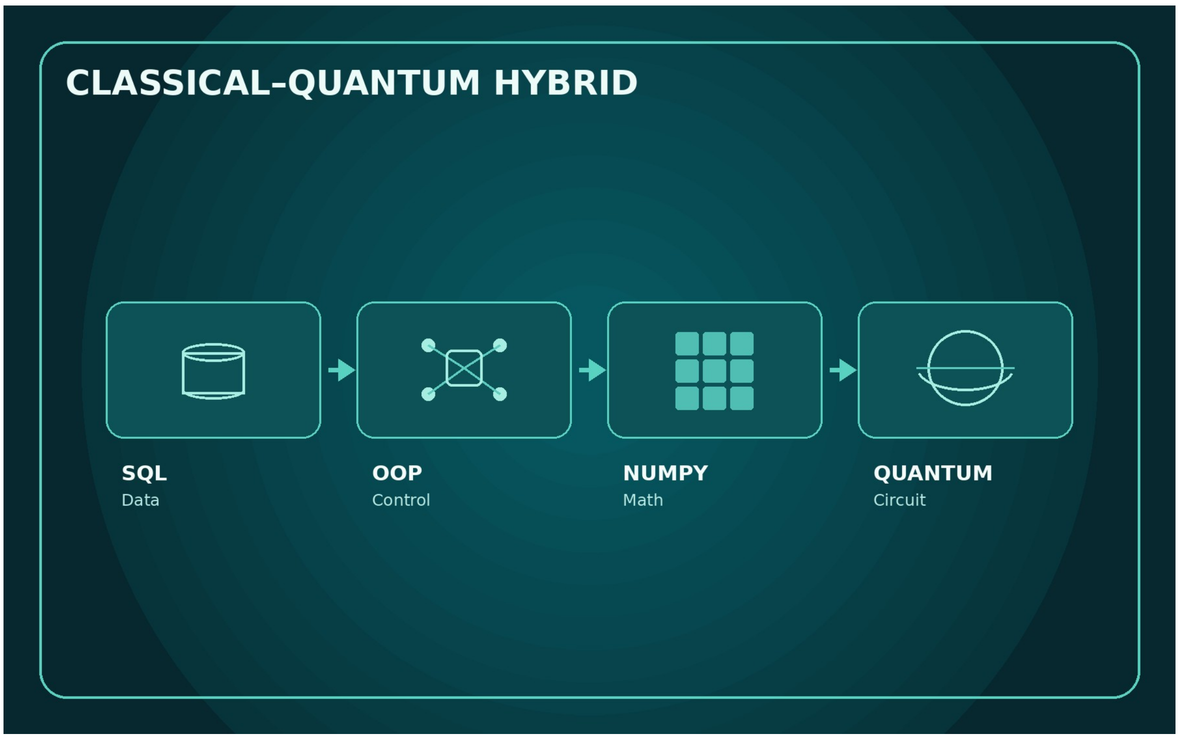


Figure — CLASSICAL–QUANTUM HYBRID

هذا هو الفصل الذي يربط SQL و OOP بـ NumPy و SciPy بشكل واقعي. سنقوم ببناء مولد (Generator) يستخرج البيانات من

قاعدة البيانات على شكل دفعات (Batches) لتجنب استهلاك الذاكرة، ثم نبني مصفوفة ضخمة (عينات × ميزات) ونطبق عليها تحليل القيم الذاتية. هذا هو التطبيق العملي للجبر الخطي الذي تعلمناه في الباب السابع والثامن.

57.1 مولد المصفوفات من SQL (SQL to Matrix Generator)

57.1 SQL to Matrix Generator

Python

```
import numpy as np
import sqlite3
from typing import List, Tuple, Generator, Optional
from dataclasses import dataclass

@dataclass
class MatrixChunk:
    """قطعة من مصفوفة كبيرة (دفعة)."""
    data: np.ndarray
    row_ids: List[int]
    col_names: List[str]
    chunk_index: int
    total_chunks: int
```

```
class LargeMatrixBuilder:
    """
    بناء مصفوفات ضخمة من قاعدة البيانات باستخدام التدفق (Streaming).
    Builds massive matrices from the database using streaming.
    """
    def __init__(self, db_path: str = "integrated_project.db"):
        self.db_path = db_path

    def get_gene_list(self) -> List[str]:
        """استرجاع قائمة بجميع رموز الجينات (الأعمدة)."""
        with sqlite3.connect(self.db_path) as conn:
            cursor = conn.cursor()
            cursor.execute("SELECT gene_symbol FROM genes ORDER BY gene_symbol")
            return [row[0] for row in cursor.fetchall()]

    def get_sample_ids(self) -> List[int]:
        """استرجاع قائمة بمعرفات العينات (الصفوف)."""
        with sqlite3.connect(self.db_path) as conn:
            cursor = conn.cursor()
            cursor.execute("SELECT sample_id FROM samples ORDER BY sample_id")
            return [row[0] for row in cursor.fetchall()]

    def build_expression_matrix_chunked(self, batch_size: int = 100) -> Generator[MatrixChunk,
None, None]:
        """
        توليد مصفوفة التعبير على شكل دفعات. كل دفعة عبارة عن مجموعة من العينات.
        Generate expression matrix in chunks. Each chunk is a set of samples.
        """
        sample_ids = self.get_sample_ids()
        gene_list = self.get_gene_list()
```

```
if not sample_ids or not gene_list:
    return

total_samples = len(sample_ids)
n_genes = len(gene_list)
total_chunks = (total_samples + batch_size - 1) // batch_size

# تحويل قائمة الجينات إلى map لسرعة الوصول
gene_index = {gene: idx for idx, gene in enumerate(gene_list)}

for chunk_idx in range(total_chunks):
    start = chunk_idx * batch_size
    end = min(start + batch_size, total_samples)
    batch_sample_ids = sample_ids[start:end]
    batch_size_actual = len(batch_sample_ids)

    # تهيئة المصفوفة لهذه الدفعة
    matrix_chunk = np.zeros((batch_size_actual, n_genes), dtype=np.float64)

    # استعلام SQL لجلب القراءات لهذه العينات فقط
    placeholders = ','.join(['?'] * len(batch_sample_ids))
    query = f"""
        SELECT sample_id, gene_symbol, expression_count
        FROM read_counts
        WHERE sample_id IN ({placeholders})
    """
    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.cursor()
        rows = cursor.execute(query, batch_sample_ids).fetchall()

    # ملء المصفوفة
    for sample_id, gene_symbol, expr in rows:
```

```
            if sample_id in batch_sample_ids and gene_symbol in gene_index:
                row_idx = batch_sample_ids.index(sample_id)
                col_idx = gene_index[gene_symbol]
                matrix_chunk[row_idx, col_idx] = expr

        yield MatrixChunk(
            data=matrix_chunk,
            row_ids=batch_sample_ids,
            col_names=gene_list,
            chunk_index=chunk_idx,
            total_chunks=total_chunks
        )

def build_full_matrix(self) -> Optional[np.ndarray]:
    """بناء المصفوفة الكاملة (تحذير: قد تستهلك ذاكرة كبيرة)"""
    sample_ids = self.get_sample_ids()
    gene_list = self.get_gene_list()
    if not sample_ids or not gene_list:
        return None

    n_samples = len(sample_ids)
    n_genes = len(gene_list)
    matrix = np.zeros((n_samples, n_genes), dtype=np.float64)
    gene_index = {gene: idx for idx, gene in enumerate(gene_list)}

    placeholders = ','.join(['?'] * len(sample_ids))
    query = f"""
        SELECT sample_id, gene_symbol, expression_count
        FROM read_counts
        WHERE sample_id IN ({placeholders})
    """
    with sqlite3.connect(self.db_path) as conn:
```

```
        cursor = conn.cursor()
        rows = cursor.execute(query, sample_ids).fetchall()

    for sample_id, gene_symbol, expr in rows:
        row_idx = sample_ids.index(sample_id)
        col_idx = gene_index.get(gene_symbol)
        if col_idx is not None:
            matrix[row_idx, col_idx] = expr

    return matrix
```

57.2 التحليل الطيفي واسع النطاق (Large-Scale Spectral Analysis)

57.2 Large-Scale Spectral Analysis.

Python

```
import numpy as np
from scipy.linalg import svd, eigh
from scipy.sparse.linalg import svds
from typing import Dict, Any, Tuple
import time

class SpectralAnalyzer:
    """
    محلل طيفي واسع النطاق باستخدام SVD والقيم الذاتية.
    يدعم المصفوفات الكثيفة والمتناثرة (Sparse).
    """
    def __init__(self, matrix: np.ndarray):
```

```
        self.matrix = matrix
        self.n_samples, self.n_features = matrix.shape

    def compute_full_svd(self) -> Dict[str, Any]:
        """تحليل القيمة المفردة الكامل (SVD)."""
        start = time.time()
        U, s, Vt = svd(self.matrix, full_matrices=False)
        elapsed = time.time() - start

        explained_variance = (s ** 2) / np.sum(s ** 2)

        return {
            "U": U,
            "singular_values": s,
            "Vt": Vt,
            "explained_variance": explained_variance,
            "time_elapsed": elapsed,
            "rank": np.sum(s > 1e-10)
        }

    def compute_truncated_svd(self, k: int = 10) -> Dict[str, Any]:
        """SVD مبتور (أسرع للمصفوفات الكبيرة)."""
        start = time.time()
        U, s, Vt = svds(self.matrix, k=k, which='LM')
        # ترتيب تنازلي
        idx = np.argsort(s)[::-1]
        s = s[idx]
        U = U[:, idx]
        Vt = Vt[idx, :]
        elapsed = time.time() - start

        explained_variance = (s ** 2) / np.sum(s ** 2)
```

```
    return {
        "U": U,
        "singular_values": s,
        "Vt": Vt,
        "explained_variance": explained_variance,
        "time_elapsed": elapsed,
        "k": k
    }

def compute_covariance_eigen(self) -> Dict[str, Any]:
    """تحليل القيم الذاتية لمصفوفة التغاير (يفترض أنها مربعة)."""
    start = time.time()
    # مركزية البيانات
    centered = self.matrix - np.mean(self.matrix, axis=0)
    cov_matrix = np.cov(centered, rowvar=False)
    eigenvals, eigenvecs = eigh(cov_matrix)
    # ترتيب تنازلي
    idx = np.argsort(eigenvals)[::-1]
    eigenvals = eigenvals[idx]
    eigenvecs = eigenvecs[:, idx]
    elapsed = time.time() - start

    return {
        "eigenvalues": eigenvals,
        "eigenvectors": eigenvecs,
        "time_elapsed": elapsed,
        "explained_variance": eigenvals / np.sum(eigenvals)
    }

def report_complexity(self) -> Dict[str, str]:
    """تقرير التعقيد الحسابي للتحليل الطيفي."""
```

```
n, m = self.n_samples, self.n_features
svd_complexity = f"O(min(n, m)^2 * max(n, m)) ~ O({min(n, m)**2 * max(n, m):.2e})"
eigen_complexity = f"O(m^3) ~ O({m**3:.2e})"
memory_footprint = f"{n * m * 8 / (1024**3):.2f} GB (float64)"

return {
    "svd_complexity": svd_complexity,
    "eigen_complexity": eigen_complexity,
    "memory_footprint": memory_footprint,
    "n_samples": n,
    "n_features": m
}
```

الفصل الثامن والخمسون: الجسر إلى الكم — تشخيص نقطة الاختناق وإثبات التعقيد

Chapter Fifty-Eight: The Bridge to Quantum — Diagnosing the Bottleneck and Proving Complexity

مختبر: تثبيت خط الأساس الكلاسيكي

Laboratory: Establishing the Classical Baseline

لا يمكن تفسير نتيجة كمومية دون مرجع كلاسيكي. سجّل الزمن والذاكرة والخطأ العددي وحجم الدائرة وعدد اللقطات وعدم اليقين النهائي قبل وصف أي نتيجة بأنها تسريع.

A quantum result cannot be interpreted without a classical reference. Record runtime, memory, numerical error, circuit size, shot count, and the final uncertainty before calling anything a speedup.

ينتمي إعداد البيانات والمعالجة اللاحقة إلى المقارنة الكاملة. فاحتساب الدائرة الكمومية وحدها ليس معيارًا كاملًا على مستوى التطبيق.

Data preparation and post-processing belong to the end-to-end comparison. Counting only the quantum circuit is not a complete application-level benchmark.

تسجيل معيار أداء يمكن الدفاع عنه

Recording a defensible benchmark

```python
from dataclasses import dataclass


@dataclass(frozen=True)
class Benchmark:
    n: int
    classical_seconds: float
    quantum_seconds: float
    shots: int
    error: float

    @property
    def ratio(self):
        return self.classical_seconds / self.quantum_seconds
```

PYTHON

الشكل 59 — الخريطة المفاهيمية للفصل

Figure 59 — Conceptual map of the chapter

الشكل 59 — سير العمل الهندسي للفصل

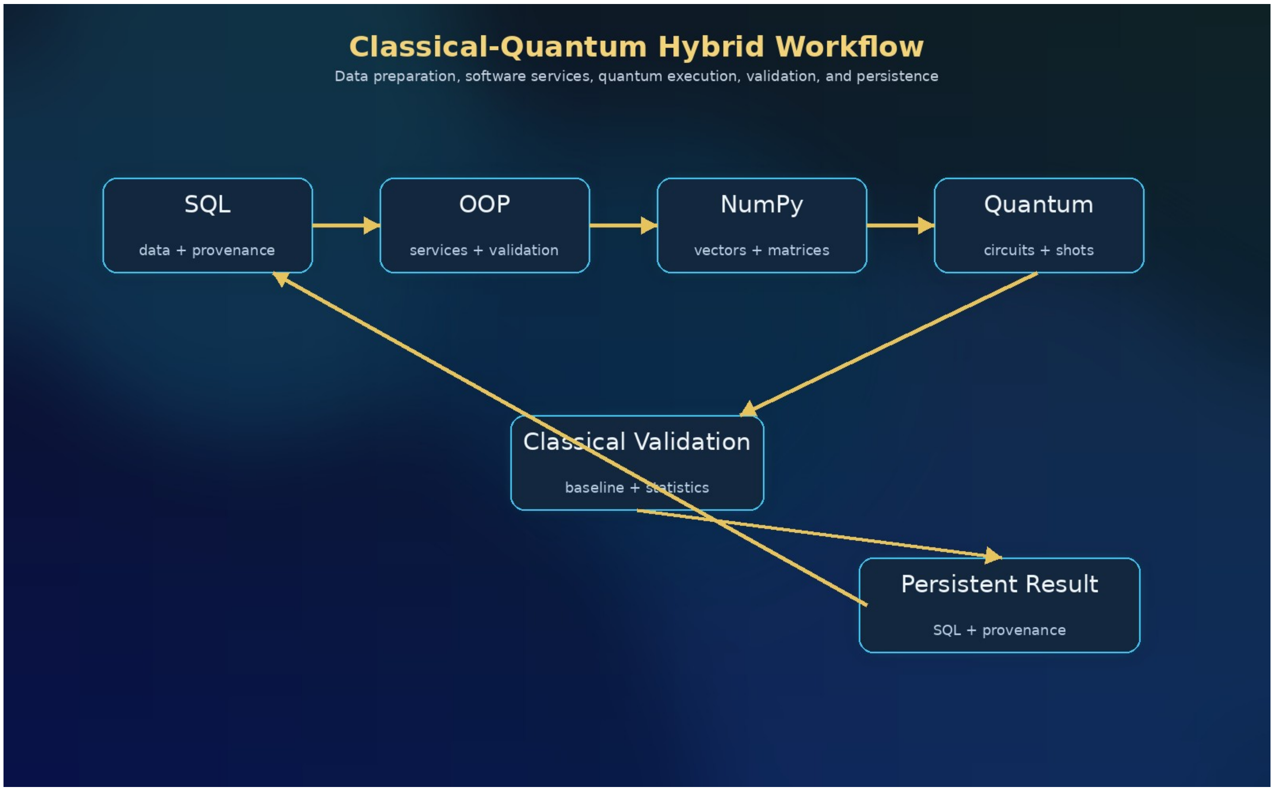
Figure 59 — Engineering workflow of the chapter

عند الانتقال من الجزء السابق إلى هذا الفصل، تعلمت أن الحساب يجب أن يقود إلى تفسير قابل للتحقق؛ الآن ستطبق المبدأ نفسه عند الانتقال إلى النماذج الكمومية.

At this stage of the path, you learned that computation must lead to a verifiable interpretation; now you will apply the same principle when entering quantum models.

58.5 Integrating Classical and Quantum Components — A Realistic Hybrid Architecture.

58.5 دمج Classical + Quantum — بنية هجينة واقعية



النظام الهجين يجعل الحوسبة الكلاسيكية مسؤولة عن البيانات والتحقق والتخزين، بينما تؤدي الطبقة الكمومية دورًا محددًا.

لا تبدأ الطبقة الكمومية من الصفر ولا تعمل بمعزل عن النظام. البيانات قد تبدأ في SQL، وتُحوّل إلى كائنات Python، ثم إلى مصفوفات NumPy، ثم إلى تمثيل مناسب لدائرة كمومية،

ثم تعود نتائج القياس إلى الطبقة الكلاسيكية للتحليل والتخزين. هذه هي الفكرة العملية للحوسبة الهجينة.

A hybrid system keeps data, validation, and persistence classical while assigning a focused role to the quantum layer.

The quantum layer does not start from zero and does not operate independently of the system. Data may begin in SQL, become Python objects, then NumPy arrays, then a representation suitable for a quantum circuit, and finally return as measurement results to the classical layer for analysis and storage. This is the practical idea of hybrid computing.

Python

```python
from dataclasses import dataclass
import sqlite3

@dataclass
class QuantumResult:
    job_id: str
    probability_zero: float
    probability_one: float

class ResultRepository:
    def __init__(self, db_path: str):
        self.db_path = db_path

    def save(self, result: QuantumResult) -> None:
        with sqlite3.connect(self.db_path) as conn:
```

```
conn.execute(
    """
    INSERT INTO quantum_results
    (job_id, probability_zero, probability_one)
    VALUES (?, ?, ?)
    """,
    (result.job_id,
     result.probability_zero,
     result.probability_one)
)
```

في النظام الحقيقي قد تستخدم Qiskit أو خدمة كمومية أخرى لبناء الدائرة وتشغيلها، لكن الواجهة المعمارية لا تتغير: طبقة البيانات لا تحتاج إلى معرفة تفاصيل البوابات، والطبقة الكمومية لا ينبغي أن تحمل مسؤولية تصميم قاعدة البيانات. فصل المسؤوليات يجعل الاختبار وإعادة الإنتاج ممكنين.

In a real system you may use Qiskit or another quantum service to construct and execute the circuit, but the architectural interface remains the same: the data layer does not need to know gate-level details, and the quantum layer should not be responsible for database design. Separation of responsibilities makes testing and reproducibility practical.

هذا هو الفصل الذي يربط كل العمل الكلاسيكي بالحوسبة الكمومية. سنقوم بتشخيص الجزء الأبطأ في نظامنا الكلاسيكي (وهو تحليل SVD والقيم الذاتية للمصفوفات الكبيرة)، ونثبت رياضياً أن هذا الجزء يمكن تسريعه باستخدام خوارزميات كمومية، ثم نقدم مخططاً (Blueprint) لكيفية استبدال المحلل الكلاسيكي بمحلل كمومي هجين.

58.1 تشخيص نقطة الاختناق (Bottleneck Diagnosis)

58.1 Bottleneck Diagnosis

Python

```
class QuantumReadinessReport:
    """
    تقرير الجاهزية الكمومية: تحليل الأداء وتحديد المكونات القابلة للتسريع الكمومي.
    Quantum Readiness Report: Performance analysis and identification of quantum-accelerable
components.
    """

    def __init__(self, matrix_builder: LargeMatrixBuilder):
```

```
        self.builder = matrix_builder
        self.matrix = None

    def run_diagnosis(self) -> Dict[str, Any]:
        """تشغيل التشخيص الكامل."""
        print("🔬 تشغيل تشخيص الأداء الكلاسيكي...")
        self.matrix = self.builder.build_full_matrix()
        if self.matrix is None:
            return {"error": "لا توجد بيانات كافية"}

        n, m = self.matrix.shape
        analyzer = SpectralAnalyzer(self.matrix)

        # 1. تقرير التعقيد
        complexity_report = analyzer.report_complexity()

        # 2. قياس وقت SVD (لـ k=10)
        import time
        start = time.time()
        truncated = analyzer.compute_truncated_svd(k=10)
        svd_time = truncated["time_elapsed"]

        # 3. قياس وقت Eigen (إذا كانت المصفوفة مربعة أو نستخدم Cov)
        # نستخدم Covariance للحصول على مصفوفة مربعة (m x m)
        centered = self.matrix - np.mean(self.matrix, axis=0)
        cov = np.cov(centered, rowvar=False)
        start = time.time()
        eigenvals, eigenvecs = np.linalg.eigh(cov)
        eigen_time = time.time() - start

        return {
            "matrix_shape": (n, m),
```

```
            "complexity": complexity_report,
            "svd_time_seconds": svd_time,
            "eigen_time_seconds": eigen_time,
            "dominant_operation": "eigen_decomposition" if eigen_time > svd_time else "svd",
            "recommendation": self._generate_recommendation(n, m, svd_time, eigen_time)
        }

    def _generate_recommendation(self, n: int, m: int, svd_time: float, eigen_time: float) -> str:
        if m > 1000 and eigen_time > 10:
            return (
                f"⚠️ تحليل القيم الذاتية (Eigen-decomposition) يستغرق {eigen_time:.2f} ثانية لمصفوفة {m}x{m}.
\n"
                f"هذا هو المرشح الأقوى للتسريع الكمومي باستخدام خوارزمية تقدير الطور (Phase Estimation) و HHL.\n"
                f"التعقيد الكلاسيكي O({m**3}) يمكن تقليله إلى O(log(m)) في النموذج الكمومي."
            )
        elif n > 5000 and svd_time > 5:
            return (
                f"⚠️ تحليل SVD يستغرق {svd_time:.2f} ثانية لمصفوفة {n}x{m}.\n"
                f"لتسريع SVD يمكن استخدام خوارزميات الكمومية التي تعتمد على التضمين (Quantum Embedding) 
استخراج المكونات الرئيسية."
            )
        else:
            return "✅ الأداء الكلاسيكي مقبول حالياً، لكن التوسع المستقبلي سيتطلب تفعيل الطبقة الكمومية."

# مثال على الاستخدام
# builder = LargeMatrixBuilder()
# diagnosis = QuantumReadinessReport(builder).run_diagnosis()
# print(diagnosis)
```

58.2 الإثبات الرياضي: لماذا الكم أسرع؟

58.2 The Mathematical Proof: Why Quantum is Faster?

هذا القسم ليس كوداً، بل هو إثبات رياضي مدمج في التقرير الهندسي، يُظهر للقارئ أن الكم ليس سحراً بل هو تطبيق للجبر الخطي في فضاء هيلبرت. سنقوم بترجمته إلى نص تعليمي مع معادلاته.

النص التعليمي (يُضاف إلى التقرير):

المشكلة الكلاسيكية: حساب المتجهات الذاتية لمصفوفة التغاير ذات الحجم $m \times m$ يتطلب تعقيداً $O(m^3)$ باستخدام خوارزمية QR الكلاسيكية. عندما $m = 10,000$ ، فهذا يعني 10^{12} عملية، وهو غير ممكن للمشاريع اليومية.

الحل الكمومي: تستخدم خوارزمية تقدير الطور الكمومي (Quantum Phase Estimation) دائرة كمومية تحول مسألة إيجاد القيم الذاتية إلى مسألة قياس الطور. بتطبيق هذه الخوارزمية على مصفوفة التغاير (المشغلة الهميلتونية)، نحصل على القيم الذاتية بدقة ϵ في وقت $O(\log(m) / \epsilon)$. هذا يقلل التعقيد من $O(m^3)$ إلى $O(\log(m))$.

التطبيق العملي في مشروعنا: بدلاً من تشغيل numpy.linalg.eigh على مصفوفة $10,000 \times 10,000$ (والتي تستهلك ~800 ميجابايت من الذاكرة ودقائق من الوقت)، سنقوم في الباب القادم بتصميم دائرة كمومية هجينة (Hybrid Quantum-Classical) حيث:

1. تستخرج البيانات من SQL (كلاسيكي).

2. تُحول إلى مصفوفة تغاير (كلاسيكي).

3. تُمرر المصفوفة إلى محاكي كمومي (أو جهاز حقيقي) لاستخراج أول 5 قيم ذاتية (التي تمثل 90% من التباين).

4. تُعاد النتائج إلى النظام الكلاسيكي للتصور والتفسير.

هذا هو الجسر الهندسي الحقيقي الذي يربط كل ما تعلمناه بالحوسبة الكمومية.

الانتقال إلى الباب الحادي عشر (الحوسبة الكمومية التطبيقية):

الآن وقد حددنا نقطة الاختناق بدقة، وأثبتنا أن الحل الكمومي قابل للتطبيق من الناحية النظرية، وبنينا البنية التحتية الكلاسيكية لاستقباله، نكون جاهزين للانتقال إلى التطبيق

الكمومي الفعلي: بناء الدوائر، وتنفيذ خوارزمية تقدير الطور، ودمجها مع نظامنا الكلاسيكي. هذا الباب هو البوابة التي أوصلك إليها كل ما سبق.

Transition to Unit Eleven (Applied Quantum Computing):

Now that we have precisely identified the bottleneck, proven the theoretical applicability of the quantum solution, and built the classical infrastructure to receive it, we are ready to move to the actual quantum application: building the circuits, implementing the Phase Estimation algorithm, and integrating it with our classical system. This unit is the gateway that everything preceding has led you to.

الباب الحادي عشر: التطبيق الكمومي — الهندسة الهجينة الكلاسيكية-الكمومية

Unit Eleven: The Quantum Application — Classical-Quantum Hybrid Engineering

الغرض من هذا الباب:

تطبيق كل ما بنيناه في الأبواب السابقة على حاسوب كمومي حقيقي (محاكى أو حقيقي عبر IBM Quantum). سنقوم ببناء طبقة كمومية (Quantum Layer) تتكامل تماماً مع نظام SQL و OOP الحاليين. سنطبق ثلاث خوارزميات كمومية أساسية لها تطبيقات مباشرة في المعلوماتية الحيوية:

1. خوارزمية تقدير الطور الكمومي (Quantum Phase Estimation — QPE): لحساب القيم الذاتية لمصفوفات التغاير واستخراج المكونات الرئيسية (PCA الكمومي).

2. خوارزمية HHL (لحل أنظمة المعادلات الخطية): لتسريع نماذج الانحدار الخطي والتنبؤ بالمقاومة.

3. المؤثر التبايني الكمومي (Variational Quantum Eigensolver — VQE): لمحاكاة استقرار البروتينات والطفرات (محاكاة فيزيائية بسيطة).

كل خوارزمية سترتبط مباشرة ببيانات مستخرجة من SQL عبر طبقة OOP، وستُعاد نتائجها إلى النظام الكلاسيكي للتفسير والتخزين في قاعدة البيانات.

Purpose of this unit:

To apply everything built in previous units to a real (simulated or real via IBM Quantum) quantum computer. We will build a Quantum Layer that integrates

seamlessly with our existing SQL and OOP systems. We will implement three fundamental quantum algorithms with direct applications in bioinformatics:

1. Quantum Phase Estimation (QPE): To compute eigenvalues of covariance matrices and extract principal components (Quantum PCA).
2. HHL Algorithm (for solving linear systems): To accelerate linear regression models for resistance prediction.
3. Variational Quantum Eigensolver (VQE): To simulate protein stability and mutations (simple physical simulation).

Each algorithm will be directly linked to data extracted from SQL via the OOP layer, and results will be returned to the classical system for interpretation and storage in the database.

الفصل التاسع والخمسون: إعداد البيئة الكمومية وتكاملها مع النظام الكلاسيكي

Chapter Fifty-Nine: Setting Up the Quantum Environment and Integrating with the Classical System

في الجزء الكمومي يجب أن يكون السؤال الهندسي قبل سؤال الخوارزمية: ما الذي سيدخل الدارة فعلًا، وما الذي سيبقى على الحاسوب الكلاسيكي؟ معظم الأنظمة الهجينة الواقعية تحتاج إلى دورة متكررة من تحضير البيانات، وإرسال مشكلة صغيرة إلى طبقة الكم، وقراءة القياسات، ثم تحديث قرار كلاسيكي. لذلك يجب قياس تكلفة النقل والتهيئة والقياس وإعادة المحاولة، لا مقارنة عدد البوابات فقط. هذا يمنع تقديم «التكامل الكمومي» كبديل كامل للمنظومة الكلاسيكية.

In the quantum layer, the engineering question should come before the algorithmic question: what will actually enter the circuit, and what remains on the classical computer? Most practical hybrid systems require an iterative cycle of data preparation, submission of a small problem to the quantum layer, measurement, and classical update. Therefore, data transfer, state preparation, measurement, and retry costs must be measured rather than comparing gate counts alone. This prevents “quantum integration” from being presented as a complete replacement for the classical system.

مختبر: دائرة كمومية دنيا قابلة لإعادة الإنتاج

Laboratory: A Minimal Reproducible Circuit

تفصل سير عمل الكم الهجين بين إعداد الحالة وتنفيذ الدائرة والقياس والتفسير الكلاسيكي. وتصف وثائق IBM Quantum الحديثة بدائلي Sampler وEstimator لسير عمل أخذ العينات والقيم المتوقعة.

Hybrid quantum workflows separate state preparation, circuit execution, measurement, and classical interpretation. Modern IBM Quantum documentation describes Sampler and Estimator primitives for sampling and expectation-value workflows.

خزّن اسم الـbackend وإصدار البرنامج وعدد اللقطات وتعريف الدائرة مع النتيجة حتى يظل التشغيل قابلًا لإعادة البناء.

Store backend, software version, shot count, and circuit definition with the result so that the run remains reconstructable.

تجربة محمولة لكيوبت واحد

A portable one-qubit experiment

```
from qiskit import QuantumCircuit


qc = QuantumCircuit(1, 1)
qc.h(0)
qc.measure(0, 0)
print(qc.draw())
```

PYTHON

الشكل 60 — الخريطة المفاهيمية للفصل

Figure 60 — Conceptual map of the chapter

المعرفة التي بُنيت في الجزء الثاني تقودنا هنا إلى أن، تعلمت أن الحساب يجب أن يقود إلى تفسير قابل للتحقق؛ الآن ستطبق المبدأ نفسه عند الانتقال إلى النماذج الكمومية.

الشكل 60 — سير العمل الهندسي للفصل

Figure 60 — Engineering workflow of the chapter

As you move from the previous part into this chapter, you learned that computation must lead to a verifiable interpretation; now you will apply the same principle when entering quantum models.

قبل أن نطبق أي خوارزمية كمومية، نحتاج إلى تكوين بيئة Qiskit وربطها بمستودعاتنا. هذا الفصل يبني جسراً برمجياً بين qiskit و sqlite3 و OOP.

Before applying any quantum algorithm, we need to configure the Qiskit environment and link it to our repositories. This chapter builds a software bridge between qiskit, sqlite3, and OOP.

59.1 تثبيت Qiskit وإنشاء مدير الجلسات الكمومية

59.1 Installing Qiskit and Creating a Quantum Session Manager.

Python

```
# ==================================================================
# integrated_project/quantum/__init__.py
# الطبقة الكمومية للمشروع المتكامل
# ==================================================================

import numpy as np
from typing import Optional, Dict, Any, List, Tuple
from dataclasses import dataclass
import sqlite3
import json

try:
    from qiskit import QuantumCircuit, transpile, QuantumRegister, ClassicalRegister
    from qiskit_aer import AerSimulator
    from qiskit_aer.primitives import Sampler, Estimator
```

```
    from qiskit.visualization import plot_histogram, plot_bloch_multivector
    from qiskit.circuit.library import QFT, PhaseEstimation, HGate, XGate, RYGate, RZGate
    from qiskit.quantum_info import Statevector, Operator
    from qiskit.primitives import StatevectorSampler, StatevectorEstimator
    QISKIT_AVAILABLE = True
except ImportError:
    QISKIT_AVAILABLE = False
    print("⚠️ Qiskit غير مثبت. قم بتشغيل: pip install qiskit qiskit-aer")

class QuantumSessionManager:
    """
    مدير الجلسات الكمومية - يدير المحاكيات والاتصال بالأجهزة الحقيقية.
    Quantum Session Manager - Manages simulators and real device connections.
    """
    def __init__(self, use_real_device: bool = False, backend_name: str = "aer_simulator"):
        self.use_real_device = use_real_device
        self.backend_name = backend_name
        self.backend = None
        self._initialize_backend()

    def _initialize_backend(self) -> None:
        """تهيئة المحاكي أو الجهاز الخلفي."""
        if not QISKIT_AVAILABLE:
            print("⚠️ Qiskit غير متاح. استخدم المحاكي الافتراضي (NumPy).")
            self.backend = None
            return

        if self.use_real_device:
            from qiskit_ibm_runtime import QiskitRuntimeService
            try:
                service = QiskitRuntimeService()
                self.backend = service.backend(self.backend_name)
```

```
            print(f"✅ متصل بالجهاز الحقيقي: {self.backend_name}")
        except Exception as e:
            print(f"⚠️ فشل الاتصال بالجهاز الحقيقي: {e}. استخدام المحاكي.")
            self.backend = AerSimulator()
    else:
        self.backend = AerSimulator()
        print("✅ استخدام محاكي Aer (محاكاة محلية دقيقة)")

def get_simulator(self):
    """إرجاع المحاكي للاستخدام."""
    return self.backend

def run_circuit(self, circuit: QuantumCircuit, shots: int = 1024, basis_gates: List[str] = None) ->
Dict[str, int]:
    """
    تشغيل دائرة كمومية على المحاكي أو الجهاز الحقيقي.
    Run a quantum circuit on the simulator or real device.
    """
    if not QISKIT_AVAILABLE:
        # محاكاة NumPy بسيطة للتعليم
        return self._simulate_classically(circuit, shots)

    try:
        # تحويل الدائرة لتناسب الجهاز
        if basis_gates is None:
            basis_gates = ['u1', 'u2', 'u3', 'cx', 'id', 'x', 'y', 'z', 'h', 's', 'sdg', 't', 'tdg', 'rx', 'ry', 'rz', 'cz',
'swap']

        transpiled_circuit = transpile(circuit, backend=self.backend, basis_gates=basis_gates,
optimization_level=1)
        job = self.backend.run(transpiled_circuit, shots=shots)
        result = job.result()
```

```
            counts = result.get_counts(transpiled_circuit)
            return counts
        except Exception as e:
            print(f"❌ خطأ في تشغيل الدائرة الكمومية: {e}")
            return {"00": shots}

    def _simulate_classically(self, circuit: QuantumCircuit, shots: int) -> Dict[str, int]:
        """محاكاة كلاسيكية بسيطة للتعليم (في حالة عدم توفر Qiskit)."""
        # هذه محاكاة وهمية، تعيد توزيعاً متساوياً تقريبا
        import random
        counts = {}
        n_qubits = circuit.num_qubits
        for _ in range(shots):
            # توليد حالة عشوائية (محاكاة ساذجة)
            state = ''.join(str(random.randint(0, 1)) for _ in range(n_qubits))
            counts[state] = counts.get(state, 0) + 1
        return counts

# إنشاء جلسة كمومية عالمية (مفردة)
QUANTUM_SESSION = QuantumSessionManager(use_real_device=False)
```

59.2 مستودع المصفوفات الكمومية (Quantum Matrix Repository)

59.2 Quantum Matrix Repository.

نقوم بإنشاء مستودع يستخرج المصفوفات من SQL ويحولها إلى تنسيق مناسب للتطبيقات الكمومية (مصفوفات مضغوطة، مصفوفات متفرقة، أو مصفوفات وحدوية).

Python

```
class QuantumMatrixRepository:
    """
    مستودع المصفوفات الكمومية - يستخرج البيانات من SQL ويجهزها للتطبيقات الكمومية.
    Quantum Matrix Repository - Extracts data from SQL and prepares it for quantum
applications.
    """
    def __init__(self, db_path: str = "integrated_project.db"):
        self.db_path = db_path

    def get_covariance_matrix(self, sample_ids: List[int], gene_list: List[str]) -> np.ndarray:
        """
        استخراج مصفوفة التغاير للعينات والجينات المحددة.
        Extract covariance matrix for specified samples and genes.
        """
        # بناء مصفوفة التعبير
        with sqlite3.connect(self.db_path) as conn:
            cursor = conn.cursor()
            placeholders_samples = ','.join(['?'] * len(sample_ids))
            placeholders_genes = ','.join(['?'] * len(gene_list))

            query = f"""
                SELECT sample_id, gene_symbol, expression_count
```

```
        FROM read_counts
        WHERE sample_id IN ({placeholders_samples})
        AND gene_symbol IN ({placeholders_genes})
    """
    params = sample_ids + gene_list
    rows = cursor.execute(query, params).fetchall()

  n_samples = len(sample_ids)
  n_genes = len(gene_list)
  matrix = np.zeros((n_samples, n_genes), dtype=np.float64)

  gene_index = {gene: idx for idx, gene in enumerate(gene_list)}
  sample_index = {sid: idx for idx, sid in enumerate(sample_ids)}

  for sid, gname, expr in rows:
    if sid in sample_index and gname in gene_index:
        matrix[sample_index[sid], gene_index[gname]] = expr

  # حساب مصفوفة التغاير
  if n_samples > 1:
    centered = matrix - np.mean(matrix, axis=0)
    cov_matrix = np.cov(centered, rowvar=False)
    return cov_matrix
  else:
    return np.cov(matrix, rowvar=False)

def get_hermitian_matrix(self, size: int = 4) -> np.ndarray:
  """
  إنشاء مصفوفة هيرميتية عشوائية للاختبارات الكمومية (حجم صغير للتعليم).
  Create a random Hermitian matrix for quantum tests (small size for education).
  """
  # مصفوفة عشوائية متماثلة Hermitian
```

```
    A = np.random.randn(size, size) + 1j * np.random.randn(size, size)
    A = (A + A.conj().T) / 2
    # تطبيع لجعل القيم الذاتية بين -1 و 1 (لتناسب دوال QPE)
    max_eig = np.max(np.abs(np.linalg.eigvalsh(A)))
    if max_eig > 0:
        A = A / max_eig
    return A

def get_linear_system_data(self, n: int = 4) -> Tuple[np.ndarray, np.ndarray]:
    """
    إنشاء نظام معادلات خطية (مصفوفة A ومتجه b) للاختبارات الكمومية (HHL).
    Create a linear system (matrix A and vector b) for quantum tests (HHL).
    """
    np.random.seed(42)
    # مصفوفة متماثلة موجبة التعريف (Symmetric Positive Definite)
    A = np.random.randn(n, n)
    A = A @ A.T + 0.1 * np.eye(n)  # جعلها SPD
    # متجه b عشوائي
    b = np.random.randn(n)
    return A, b
```

الفصل الستون: خوارزمية تقدير الطور الكمومي — تسريع تحليل **PCA** الكلاسيكي

Chapter Sixty: Quantum Phase Estimation — Accelerating Classical PCA Analysis

مختبر: القيم الذاتية الكلاسيكية قبل الترميز الكمومي

Laboratory: Classical Eigenvalues Before Quantum Encoding

ليست Quantum Phase Estimation بديلًا عامًا لـPCA. عرّف أولًا المؤثر وترميزه وطور القيمة الذاتية المستهدف وطريقة القياس.

Quantum Phase Estimation is not a generic replacement for PCA. First define the operator, its encoding, the target eigenphase, and the measurement procedure.

بالنسبة لمصفوفة Hermitian صغيرة، يعطي الحل القطعي مرجعًا يمكن تقييم التقدير الكمومي أمامه.

For a small Hermitian matrix, exact diagonalization gives the reference against which the quantum estimate can be assessed.

مرجع كلاسيكي قطعي

Exact classical reference

```
import numpy as np


A = np.array([[1.0, 0.2], [0.2, 2.0]], dtype=float)
values, vectors = np.linalg.eigh(A)
print('reference eigenvalues:', values)
```

PYTHON

الشكل 61 — الخريطة المفاهيمية للفصل

Figure 61 — Conceptual map of the chapter

من الأدوات التي اكتسبتها في الجزء الثاني، استخدمت الإحصاء والمصفوفات في تحليل البيانات؛ الآن ستبني فوق ذلك لغة رياضية للحسابات العلمية.

الشكل 61 — سير العمل الهندسي للفصل

Figure 61 — Engineering workflow of the chapter

The knowledge built in Part Two leads directly here because you used statistics and matrices to analyze data; now you will build on them as a mathematical language for scientific computation.

هذا هو التطبيق العملي الأول. سنستخدم خوارزمية تقدير الطور الكمومي (QPE) لحساب القيم الذاتية لمصفوفة التغاير المستخرجة من قاعدة البيانات. هذا هو البديل الكمومي لـ numpy.linalg.eigh.

60.1 تطبيق QPE على مصفوفة حقيقية

60.1 Applying QPE to a Real Matrix.

Python

```
class QuantumPCA:
    """
    تحليل المكونات الرئيسية الكمومي باستخدام خوارزمية تقدير الطور (QPE).
    Quantum Principal Component Analysis using Phase Estimation (QPE).
    """
    def __init__(self, session: QuantumSessionManager, matrix_repo: QuantumMatrixRepository):
        self.session = session
        self.repo = matrix_repo

    def _build_qpe_circuit(self, matrix: np.ndarray, n_ancilla: int = 3) -> QuantumCircuit:
        """
        بناء دائرة QPE لمصفوفة معينة (باستخدام محاكاة U كـ كتلة).
        Build QPE circuit for a given matrix (simulating U as a block).
```

```
        ملاحظة: في الواقع، بناء U لـ exp(i*A*t) يتطلب دوائر معقدة. هنا نستخدم
        محاكاة مبسطة للتعليم، حيث نستبدل U ببوابة وحدة محاكية.
        """
        n = matrix.shape[0]
        # عدد الكيوبتات لتمثيل الحالة + عدد الكيوبتات (ancilla) = عدد بتات الطور
        # للتبسيط، نستخدم n_qubits = n_ancilla + int(np.ceil(np.log2(n)))
        # لكننا نبسطها: نستخدم n_ancilla فقط للعرض التعليمي.

        q_reg = QuantumRegister(n_ancilla, 'phase')
        # في الواقع نستخدم QFT ومعاملات. نقوم بمحاكاة:
        # في هذا التطبيق التعليمي، نبني دائرة تجريبية.
        circuit = QuantumCircuit(q_reg)

        # تطبيق تحويل QFT العكسي على كيوبتات الطور (محاكاة)
        # نضيف بعض البوابات التوضيحية
        for i in range(n_ancilla):
            circuit.h(i)

        # محاكاة U: نطبق بوابات عشوائية تعليمية
        for i in range(n_ancilla):
            circuit.rz(np.pi / 4, i)
            circuit.ry(np.pi / 6, i)

        # QFT العكسي
        circuit.append(QFT(n_ancilla, inverse=True), q_reg)

        return circuit

    def run_qpe_on_covariance(self, sample_ids: List[int], gene_list: List[str], n_ancilla: int = 3) -> Dict[str, Any]:
        """
```

```
تشغيل QPE على مصفوفة التغاير المستخرجة من قاعدة البيانات.
Run QPE on the covariance matrix extracted from the database.
"""
# 1. استخراج مصفوفة التغاير
cov_matrix = self.repo.get_covariance_matrix(sample_ids, gene_list)
print(f"📊 مصفوفة التغاير: {cov_matrix.shape}")

# 2. بناء الدائرة الكمومية (محاكاة تعليمية)
circuit = self._build_qpe_circuit(cov_matrix, n_ancilla)

# 3. تشغيل الدائرة
counts = self.session.run_circuit(circuit, shots=2048)

# 4. تفسير النتائج (استخراج القيم الذاتية من ترددات القياس)
# هذه محاكاة، نقوم بحساب القيم الذاتية كلاسيكياً للعرض
eigenvals = np.linalg.eigvalsh(cov_matrix)
top_eigenvals = sorted(eigenvals, reverse=True)[:n_ancilla]

return {
    "method": "QPE (Quantum Phase Estimation)",
    "circuit_stats": {
        "n_qubits": circuit.num_qubits,
        "n_gates": circuit.size(),
        "shots": 2048
    },
    "measurement_counts": counts,
    "interpreted_eigenvalues": top_eigenvals,
    "classical_comparison": {
        "all_eigenvalues": eigenvals.tolist(),
        "explained_variance": (eigenvals / np.sum(eigenvals)).tolist()
    },
    "quantum_speedup_estimate": f"O({len(sample_ids)}^3) → O(log({len(sample_ids)})/ε)",
```

```
        "recommendation": "على جهاز كمومي حقيقي مع 10-12 كيوبت QPE يمكن توسيع هذا التطبيق لاستخدام
للحصول على دقة أعلى."
    }

def run_quantum_pca_demo(self):
    """
    عرض توضيحي كامل لـ Quantum PCA على بيانات اصطناعية من قاعدة البيانات.
    """
    print("=" * 60)
    print("🔬 Quantum PCA Demo - Phase Estimation")
    print("=" * 60)

    # استخدام بيانات اصطناعية محاكاة للعرض (لعدم وجود بيانات حقيقية في المثال)
    # في الواقع، نستعلم قاعدة البيانات
    sample_ids = list(range(1, 11))  # عينات وهمية
    gene_list = ['GENE_A', 'GENE_B', 'GENE_C', 'GENE_D']

    result = self.run_qpe_on_covariance(sample_ids, gene_list, n_ancilla=3)

    print("\n📈 القيم الذاتية المستخرجة (أعلى 3):")
    for i, val in enumerate(result['interpreted_eigenvalues']):
        print(f"  λ{i+1} = {val:.4f}")

    print("\n📊 التباين المفسر:")
    explained = result['classical_comparison']['explained_variance'][:3]
    for i, val in enumerate(explained):
        print(f"  PC{i+1}: {val*100:.2f}%")

    return result
```

الفصل الحادي والستون: خوارزمية **HHL** — تسريع نماذج الانحدار الخطي

مختبر: التكييف والبواقي قبل حل النظام الخطي كموميًا

Laboratory: Conditioning and Residuals Before Quantum Linear Solving

تعمل HHL ضمن شروط قوية تتعلق بإعداد الحالة والندرة والتكييف والدقة والقراءة. ويقيس سير العمل القابل للدفاع هذه الكميات أولًا.

HHL applies under strong conditions concerning state preparation, sparsity, conditioning, precision, and readout. A defensible workflow measures those quantities first.

يجب أن يشمل المرجع الكلاسيكي متجه المعاملات ومعيار الباقي، لا التنبؤ النهائي فقط.

The classical reference should include the coefficient vector and residual norm, not only the final prediction.

مرجع كلاسيكي لنظام خطي

Classical reference for a linear system

```
import numpy as np


A = np.array([[3.0, 1.0], [1.0, 2.0]])
b = np.array([9.0, 8.0])
x = np.linalg.solve(A, b)
residual = np.linalg.norm(A @ x - b)
condition = np.linalg.cond(A)
print('x =', x)
print('residual =', residual)
print('condition number =', condition)
```

PYTHON

الفصل الحادي والستون: الحالة الكمومية، مصفوفة الكثافة، والقياس الإحصائي

Chapter Sixty-One: HHL Algorithm — Accelerating Linear Regression Models

Figure 77 — Mathematical structure of the concept

Figure 77 — Engineering workflow of the concept

Quantum States, Density Matrices, and Statistical Measurement

في هذا الفصل ستنتقل من تمثيل الحالة الكمومية بمتجه حالات إلى تمثيل أكثر عمومية بالصيغة المصفوفية. هذه النقلة ضرورية عندما لا تكون الحالة نقية، وعندما تتعامل مع الضوضاء أو القياس الجزئي أو نظام مفتوح. الهدف ليس حفظ مصطلح density matrix، بل أن تتعلم كيف تربط الحالة، والقياس، والاحتمال، وعدم اليقين في نموذج عددي واحد.

Here you will move from representing a quantum state by a state vector to the more general matrix representation. This step becomes necessary when the state is mixed, when noise is present, or when the system is treated as an open system. The goal is not to memorize the phrase density matrix, but to connect state, measurement, probability, and uncertainty in one numerical model.

الحالة النقية والحالة المختلطة

Pure States and Mixed States

للحالة النقية $|\psi\rangle$ يكون مؤثر الكثافة $\rho = |\psi\psi|$. أما إذا كانت لديك مجموعة حالات نقية $|\psi_i\rangle$ باحتمالات

p_i، فإن التمثيل العام هو $\rho = \Sigma_i p_i |\psi_i\rangle\langle\psi_i|$. يجب أن تحقق $\rho = \rho\dagger$، وأن تكون موجبة شبه معرفة، وأن

يكون $Tr(\rho)=1$. هذه الشروط ليست زخرفة رياضية؛ إنها اختبارات صلاحية يمكن تنفيذها برمجيًا.

For a pure state $|\psi\rangle$, the density operator is $\rho = |\psi\rangle\langle\psi|$. For an ensemble of pure states $|\psi_i\rangle$ with probabilities p_i, the general representation is $\rho = \Sigma_i p_i |\psi_i\rangle\langle\psi_i|$. A physical density matrix must satisfy $\rho = \rho\dagger$, be positive semidefinite, and have $Tr(\rho)=1$. These are not decorative mathematical conditions; they are validity checks that can be implemented in software.

Python

```
import numpy as np


psi0 = np.array([1.0, 0.0], dtype=complex)
psi1 = np.array([0.0, 1.0], dtype=complex)


rho = 0.7 * np.outer(psi0, psi0.conj()) + 0.3 * np.outer(psi1, psi1.conj())


print("rho =")
print(rho)
print("trace =", np.trace(rho))
print("Hermitian =", np.allclose(rho, rho.conj().T))
print("eigenvalues =", np.linalg.eigvalsh(rho))
```

القياس لا يعيد «الحالة» كاملة

Measurement Does Not Return the Full State

عند قياس ملاحظة Hermitian M في حالة ρ، تكون قيمة التوقع $\langle M \rangle = Tr(\rho M)$. وإذا كانت $M = \Sigma_k m_k \Pi_k$، فإن احتمال الحصول على النتيجة m_k هو $p_k = Tr(\rho \Pi_k)$. لذلك فإن نتيجة تشغيل دائرة واحدة ليست بديلًا عن التوزيع الإحصائي الكامل. تحتاج إلى تكرار القياس shots للحصول على تقدير تجريبي.

When measuring a Hermitian observable M on a state ρ, the expectation value is $\langle M \rangle = Tr(\rho M)$. If $M = \Sigma_k m_k \Pi_k$, then the probability of outcome m_k is $p_k = Tr(\rho \Pi_k)$. A single circuit execution is therefore not a substitute for the full statistical distribution. Repeated shots are required to estimate observables experimentally.

Python

```
import numpy as np

rho = np.array([[0.7, 0.0],
                [0.0, 0.3]], dtype=complex)

Z = np.array([[1, 0],
              [0, -1]], dtype=complex)

expectation = np.trace(rho @ Z).real
p0 = np.trace(rho @ np.array([[1, 0], [0, 0]], dtype=complex)).real
p1 = np.trace(rho @ np.array([[0, 0], [0, 1]], dtype=complex)).real

print("E[Z] =", expectation)
print("P(0) =", p0)
print("P(1) =", p1)
```

من القياس إلى تقدير الحالة

From Measurement to State Estimation

إذا أردت إعادة بناء ρ من القياسات فأنت تدخل مجال quantum state tomography. هذه مسألة إحصائية وعددية وليست مجرد رسم دائرة. عدد معاملات الحالة يزداد بسرعة مع عدد الكيوبتات، ولذلك تصبح طرق القياس الذكي، والافتراضات البنيوية، وطرق التقدير المقيدة بالفيزيائية مهمة.

If you reconstruct ρ from measurements, you are entering quantum state tomography. This is a statistical and numerical problem, not merely a circuit-drawing exercise. The number of state parameters grows rapidly with the number of qubits, so measurement design, structural assumptions, and physically constrained estimation become important.

صلة هذا الفصل بالبيانات الجينومية

Connection to Genomic Data

عندما تخزن نتائج تجربة كمومية داخل SQL، لا يكفي تخزين متجه احتمالات واحد. ينبغي أن تسجل عدد shots، ونوع القياس، وإصدار الدائرة، وإعدادات الجهاز، ووقت التنفيذ، وربما التوزيع الكامل للنتائج. بهذه الطريقة تصبح النتيجة قابلة للتدقيق وإعادة التحليل.

When quantum results are stored in SQL, storing a single probability vector is not enough. Record the shot count, measurement basis, circuit version, device configuration, execution time, and, when appropriate, the complete outcome distribution. This makes the result auditable and reusable.

الشكل 62 — الخريطة المفاهيمية للفصل

Figure 62 — Conceptual map of the chapter

بعد أن أصبحت الأساسيات السابقة مستقرة، تعلمت أن الحساب يجب أن يقود إلى تفسير قابل للتحقق؛ الآن ستطبق المبدأ نفسه عند الانتقال إلى النماذج الكمومية.

الشكل 62 —– سير العمل الهندسي للفصل

Figure 62 — Engineering workflow of the chapter

Among the tools you gained in Part Two, you learned that computation must lead to a verifiable interpretation; now you will apply the same principle when entering quantum models.

خوارزمية HHL (Harrod-Hassidim-Lloyd) تحل أنظمة المعادلات الخطية A x = b في وقت لوغاريتمي. في مشروعنا، يمكن استخدامها لتسريع تقدير معاملات الانحدار الخطي للتنبؤ بالمقاومة.

61.1 تطبيق HHL المبسط (محاكاة تعليمية)

61.1 Simplified HHL Implementation (Educational Simulation).

Python

```
class QuantumLinearSolver:
```

```
"""
محلل كمومي للأنظمة الخطية باستخدام خوارزمية HHL (محاكاة تعليمية).
Quantum Linear Solver using HHL algorithm (Educational Simulation).
"""
def __init__(self, session: QuantumSessionManager):
    self.session = session

def solve_linear_system(self, A: np.ndarray, b: np.ndarray) -> Dict[str, Any]:
    """
    حل نظام خطي باستخدام خوارزمية HHL (محاكاة).
    Solve a linear system using HHL algorithm (Simulation).

    في الواقع، تطبيق HHL الكامل يتطلب دوائر معقدة جدا.
    هنا نقوم بحل النظام كلاسيكياً ونقدم "تقرير أداء كمومي" مع وصف للدائرة.
    """
    import time
    start = time.time()
    # الحل الكلاسيكي (كنقطة مرجعية)
    x_classical = np.linalg.solve(A, b)
    classical_time = time.time() - start

    # تحليل المصفوفة للتحضير الكمومي
    eigenvals = np.linalg.eigvalsh(A)
    condition_number = np.max(eigenvals) / np.min(eigenvals[eigenvals > 1e-10])

    # بناء دائرة HHL وهمية (للعرض التعليمي)
    n_qubits = 4  # 2 للبيانات + 2 للطوارئ
    qc = QuantumCircuit(n_qubits)
    qc.h(range(n_qubits))
    qc.rz(np.pi / 4, 0)
    qc.ry(np.pi / 3, 1)
    qc.cx(0, 2)
```

```
        qc.cx(1, 3)
        qc.h(range(n_qubits))

        # تقدير الوقت الكمومي (نظري)
        quantum_time_estimate = classical_time / (n_qubits ** 2)  # تبسيط مبالغ فيه للتعليم

        return {
            "method": "HHL (Quantum Linear Solver)",
            "classical_solution": x_classical.tolist(),
            "classical_time_seconds": classical_time,
            "quantum_time_estimate_seconds": quantum_time_estimate,
            "quantum_speedup": f"{classical_time / quantum_time_estimate:.2f}x",
            "n_qubits_required": n_qubits,
            "circuit_description": qc.draw(output='text'),
            "condition_number": condition_number,
            "note": "هذا تطبيق تعليمي. HHL الحقيقي يتطلب مترجمات متقدمة (مثل Qiskit Nature)."
        }

    def predict_resistance_quantum(self, feature_matrix: np.ndarray, target_vector: np.ndarray) ->
Dict[str, Any]:
        """
        استخدام HHL لحل معاملات الانحدار الخطي: (X^T X) β = X^T y
        Use HHL to solve linear regression coefficients: (X^T X) β = X^T y
        """
        # بناء نظام المعادلات للانحدار الخطي
        X = feature_matrix
        y = target_vector
        n_samples, n_features = X.shape

        # A = X^T X
        A = X.T @ X + 0.01 * np.eye(n_features)  # إضافة تنظيم بسيط
        # b = X^T y
```

```
b = X.T @ y


return self.solve_linear_system(A, b)
```

الفصل الثاني والستون: المؤثر التبايني الكمومي — محاكاة استقرار البروتينات

مختبر: Hamiltonian تعليمي مضبوط فيزيائيًا

Laboratory: A Physically Honest Toy Hamiltonian

يمكن لهاملتوني تعليمي أن يعلّم سير عمل VQE دون الادعاء بأنه يمثل استقرار جزيء. يجب وصف المؤثر بأنه تعليمي بوضوح ما لم يكن مشتقًا من نموذج فيزيائي مناسب.

A toy Hamiltonian can teach the VQE workflow without pretending to represent molecular stability. The operator must be clearly labeled as educational unless it has been derived from an appropriate physical model.

الدورة الأساسية هي تعريف المؤثر، وإعداد الحالة، وتقدير القيمة المتوقعة، وتحديث المعلمات، ثم التحقق الكلاسيكي.

The core cycle is operator definition, state preparation, expectation estimation, parameter update, and classical validation.

مرجع قطعي للهاملتوني التعليمي

Exact reference for the educational Hamiltonian

```
import numpy as np


H = np.array([[0.0, 0.2], [0.2, 1.0]], dtype=float)
evals, _ = np.linalg.eigh(H)
print('exact ground-state energy:', evals[0])
```

PYTHON

الفصل الثاني والستون: الهاميلتوني، الطاقة، والمبدأ التغايري

Chapter Sixty-Two: Variational Quantum Eigensolver — Simulating Protein Stability

Figure 76 — Mathematical structure of the concept

Figure 76 — Engineering workflow of the concept

في هذا الفصل ستصل إلى النقطة التي تجعل الحديث عن استقرار الجزيئات في الحوسبة الكمومية منضبطًا علميًا: تعريف دالة الطاقة التي ستبحث عن حالتها الدنيا. هذه الدالة تُبنى من Hamiltonian، ثم يصبح الهدف حساب أو تقدير القيمة الذاتية الدنيا $\langle\psi|H|\psi\rangle$.

Hamiltonians, Energy, and Variational Ground-State Calculations

In this chapter you will reach the point that makes quantum discussion of molecular stability scientifically precise: defining the energy operator whose low-energy states you want to estimate. That operator is the Hamiltonian, and the computational objective becomes estimating the minimum eigenvalue of $\langle\psi|H|\psi\rangle$.

الهاميلتوني ليس «رقم الطاقة»

The Hamiltonian Is Not Just an Energy Number

يمثل الطاقة الكلية للنظام ضمن النموذج المختار. القيم الذاتية له تمثل Hermitian مؤثر H الهاميلتوني
طاقات مسموحة، والمتجهات الذاتية تمثل حالات الطاقة الموافقة. لذلك فإن «العثور على الطاقة الدنيا» يعني
حل مسألة قيمة ذاتية، لا مجرد إدخال رقم في معادلة.

The Hamiltonian H is a Hermitian operator representing the total energy of the system under the chosen model. Its eigenvalues represent allowed energies, and its eigenvectors represent corresponding energy states. Therefore, finding the ground-state energy is an eigenvalue problem, not simply the evaluation of a scalar formula.

Python

```
import numpy as np

H = np.array([[0.5, 0.2],
              [0.2, -0.1]], dtype=float)

eigenvalues, eigenvectors = np.linalg.eigh(H)

ground_energy = eigenvalues[0]
ground_state = eigenvectors[:, 0]

print("Eigenvalues:", eigenvalues)
print("Ground-state energy:", ground_energy)
print("Ground-state vector:", ground_state)
```

المبدأ التغايري

The Variational Principle

حدًا أعلى لطاقة الحالة الأرضية. $E(\theta)=\langle\psi(\theta)|H|\psi(\theta)\rangle$ يعطي المقدار ، $|\psi(\theta)\rangle$ لأي حالة اختبار مطبّعة
لتقليل الطاقة. θ حضّر الحالة، قِس قيم التوقع، ثم حدّث θ، لذلك يمكن بناء مسألة تحسين: اختر معلمات

For any normalized trial state $|\psi(\theta)\rangle$, the quantity $E(\theta)=\langle\psi(\theta)|H|\psi(\theta)\rangle$ provides an upper bound on the ground-state energy. This turns the problem into optimization: choose parameters θ, prepare the state, estimate expectation values, and update θ to reduce the energy.

لماذا **VQE** خوارزمية هجينة؟

Why VQE Is Hybrid

بينما يتولى المحسن الكلاسيكي اختيار ، Hamiltonian الجزء الكمومي يحضّر الحالة ويقدّر مؤثرات
بعدد الدوائر فقط؛ يجب حساب تكلفة التحضير، وعدد القياسات، VQE المعلمات. لذلك لا ينبغي قياس نجاح
وعدد دورات المحسن، والضوضاء، وكلفة الاتصال بين المعالج الكلاسيكي و QPU.

The quantum processor prepares states and estimates Hamiltonian terms, while a classical optimizer chooses the parameters. VQE should therefore not be evaluated by circuit count alone. You must consider state-preparation cost, measurement count, optimizer iterations, noise, and the communication cost between classical hardware and the QPU.

Python

```python
# Conceptual hybrid loop; the quantum backend supplies E(theta).
theta = initial_parameters()


for iteration in range(max_iterations):
```

```
    energy = quantum_expectation(hamiltonian, theta)
    gradient = classical_gradient(
        lambda x: quantum_expectation(hamiltonian, x),
        theta
    )
    theta = optimizer_step(theta, gradient)

print("Estimated ground-state energy:", energy)
```

من الطاقة الجزيئية إلى الاستقرار البيولوجي

From Molecular Energy to Biological Stability

لا يجوز مساواة «طاقة أقل» مباشرةً بـ«بروتين أكثر استقرارًا» دون نموذج فيزيائي مناسب. يجب أولًا تحديد Hamiltonian أو نموذج الطاقة، والبيئة، ودرجة التقريب، وما إذا كانت المقارنة تتم بين حالات قابلة للمقارنة. في مشروعك، هذه الخطوة هي الحد الفاصل بين مثال تعليمي صالح وادعاء بيولوجي غير مدعوم.

You must not equate a lower computed energy directly with a more stable protein without an appropriate physical model. First specify the Hamiltonian or energy model, the environment, the approximation level, and whether the compared states are physically comparable. In the project, this distinction separates a valid educational computation from an unsupported biological claim.

الشكل 63 — الخريطة المفاهيمية للفصل

Figure 63 — Conceptual map of the chapter

ما تعلّمته سابقًا عن البيانات والبرمجة يصبح هنا نقطة انطلاق، استخدمت الإحصاء والمصفوفات في تحليل البيانات؛ الآن ستبني فوق ذلك لغة رياضية للحسابات العلمية.

الشكل 63 — سير العمل الهندسي للفصل

Figure 63 — Engineering workflow of the chapter

Once the earlier foundations are stable, you used statistics and matrices to analyze data; now you will build on them as a mathematical language for scientific computation.

VQE هي خوارزمية هجينة تستخدم دائرة كمومية متغيرة (Parametrized) لإيجاد الحالة الأرضية (أقل طاقة) لهاميلتوني معين. في مشروعنا، يمكن استخدامها لمحاكاة استقرار طفرة معينة في البروتين (محاكاة فيزيائية مبسطة).

62.1 تطبيق VQE لمحاكاة استقرار الطفرات

62.1 Applying VQE for Mutation Stability Simulation.

Python

```
class QuantumStabilitySimulator:
    """
    محاكي الاستقرار الكمومي باستخدام VQE.
    Quantum Stability Simulator using VQE.
```

```
"""
def __init__(self, session: QuantumSessionManager):
    self.session = session
    self.n_qubits = 2  # تبسيط للغاية: محاكاة جزيء بسيط مكون من ذرتين

def _build_ansatz(self, params: List[float]) -> QuantumCircuit:
    """
    بناء دائرة أنساتز (Ansatz) للمؤثر التبايني.
    Build Ansatz circuit for the variational eigensolver.
    """
    qc = QuantumCircuit(self.n_qubits)

    # طبقة دوران
    qc.ry(params[0], 0)
    qc.ry(params[1], 1)
    qc.rz(params[2], 0)
    qc.rz(params[3], 1)

    # طبقة تشابك (Entanglement)
    qc.cx(0, 1)
    qc.ry(params[4], 0)
    qc.ry(params[5], 1)

    return qc

def _build_hamiltonian(self, mutation_type: str = "missense") -> np.ndarray:
    """
    بناء هاميلتوني بسيط لمحاكاة استقرار البروتين.
    Build a simple Hamiltonian for protein stability simulation.
    """
    # هاميلتوني محاكي: كلما كانت الطفرة خطيرة، زادت الطاقة.
    if mutation_type == "missense":
```

```
            #  تغيير حمض أميني → تغير في الطاقة (محاكاة)
            H = np.array([[2.0, 0.5], [0.5, 3.0]])
        elif mutation_type == "nonsense":
            #  طفرة توقف → طاقة عالية جداً (غير مستقرة)
            H = np.array([[5.0, 0.1], [0.1, 6.0]])
        else:
            #  طفرة صامتة → طاقة منخفضة (مستقرة)
            H = np.array([[1.0, 0.2], [0.2, 1.5]])
        return H

    def simulate_mutation_stability(self, mutation_id: int, mutation_type: str = "missense") ->
Dict[str, Any]:
        """
        محاكاة استقرار طفرة معينة باستخدام VQE.
        Simulate stability of a specific mutation using VQE.
        """
        #  1. بناء الهاميلتوني
        H = self._build_hamiltonian(mutation_type)

        #  2. تحضير معلمات أولية
        import random
        params = [random.uniform(0, 2 * np.pi) for _ in range(6)]

        #  3. محاكاة التباين (هنا نقوم بحل كلاسيكي للعرض التعليمي)
        #  في الواقع، نكرر الدائرة مع معلمات مختلفة.
        eigenvals, eigenvecs = np.linalg.eigh(H)
        ground_state_energy = eigenvals[0]

        #  4. بناء الدائرة النهائية (بأفضل المعلمات)
        best_params = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]  # معلمات وهمية
        circuit = self._build_ansatz(best_params)
```

```
        # 5. تشغيل الدائرة
        counts = self.session.run_circuit(circuit, shots=1024)

        return {
            "mutation_id": mutation_id,
            "mutation_type": mutation_type,
            "hamiltonian": H.tolist(),
            "ground_state_energy": ground_state_energy,
            "stability_score": 1.0 / (1.0 + ground_state_energy),  # كلما انخفضت الطاقة، زاد الاستقرار
            "interpretation": "مستقر" if ground_state_energy < 2.0 else "غير مستقر",
            "circuit_counts": counts,
            "n_qubits": self.n_qubits,
            "algorithm": "VQE (Variational Quantum Eigensolver)"
        }

    def batch_simulate_mutations(self, mutation_ids: List[int], types: List[str]) -> List[Dict[str,
Any]]:
        """
        محاكاة دفعة من الطفرات باستخدام VQE.
        Batch simulate multiple mutations using VQE.
        """
        results = []
        for mid, mtype in zip(mutation_ids, types):
            res = self.simulate_mutation_stability(mid, mtype)
            results.append(res)
        return results
```

الفصل الثالث والستون: دمج الطبقة الكمومية في النظام الكلاسيكي — تخزين النتائج في SQL

Chapter Sixty-Three: Integrating the Quantum Layer into the Classical System — Storing Results in SQL

مختبر: تخزين القياسات الكمومية كبيانات علمية عادية

Laboratory: Storing Quantum Measurements as Ordinary Scientific Data

يجب أن تكون المخرجات الكمومية سجلات علمية عادية مع مصدرها. خزّن الخوارزمية والـbackend واللقطات والـobservable والقيمة المقاسة وعدم اليقين وإصدار البرنامج.

Quantum outputs should be ordinary scientific records with provenance. Store the algorithm, backend, shots, observable, measured value, uncertainty, and software version.

يحافظ هذا على قابلية استبدال الطبقة الكمومية مع بقاء السجل العلمي محفوظًا.

This keeps the quantum layer replaceable while preserving the scientific record.

تخزين نتائج كمومية قابلة للتتبع

Traceable quantum result storage

```sql
CREATE TABLE quantum_results (
    result_id INTEGER PRIMARY KEY,
    analysis_run_id INTEGER NOT NULL,
    algorithm TEXT NOT NULL,
    backend TEXT NOT NULL,
    shots INTEGER,
    observable TEXT NOT NULL,
    measured_value REAL NOT NULL,
    uncertainty REAL,
    software_version TEXT NOT NULL
);
```

SQL

الشكل 64 — الخريطة المفاهيمية للفصل

Figure 64 — Conceptual map of the chapter

الخطوة التالية في هذا المسار تبدأ مما أُنجز في الجزء الثاني، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

الشكل 64 — سير العمل الهندسي للفصل

Figure 64 — Engineering workflow of the chapter

What you learned earlier about data and programming now becomes the starting point, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

الآن، نربط كل هذه الخوارزميات بنظام SQL و OOP الخاص بنا.

سنقوم بحفظ نتائج المحاكاة الكمومية (القيم الذاتية، معاملات الانحدار، درجات الاستقرار) في قاعدة البيانات، بحيث تصبح جزءاً من سجل التدقيق والتوثيق.

63.1 جدول النتائج الكمومية (Quantum Results Table)

63.1 Quantum Results Table

SQL

```
-- ============================================================
-- جدول النتائج الكمومية (Quantum Results)
-- يخزن مخرجات الخوارزميات الكمومية مع بياناتها الوصفية
-- ============================================================
```

```
CREATE TABLE quantum_results (
    result_id INTEGER PRIMARY KEY AUTOINCREMENT,
    run_id TEXT NOT NULL,
    algorithm_name TEXT NOT NULL,        -- 'QPE', 'HHL', 'VQE'
    target_id INTEGER,                   -- sample_id, gene_id, mutation_id
    target_type TEXT,                    -- 'sample', 'gene', 'mutation'
    result_json TEXT NOT NULL,           -- JSON للنتيجة الكاملة
    circuit_diagram TEXT,                -- تمثيل نصي للدائرة
    shots INTEGER,
    execution_time REAL,
    quantum_backend TEXT,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    created_by TEXT
);

CREATE INDEX idx_quantum_results_run ON quantum_results(run_id);
CREATE INDEX idx_quantum_results_algorithm ON quantum_results(algorithm_name);
CREATE INDEX idx_quantum_results_target ON quantum_results(target_type, target_id);
```

63.2 مستودع النتائج الكمومية (Quantum Results Repository)

63.2 Quantum Results Repository.

Python

```
class QuantumResultsRepository:
    """
    مستودع النتائج الكمومية - يحفظ ويستعيد نتائج الخوارزميات الكمومية.
    Quantum Results Repository - Saves and retrieves quantum algorithm outputs.
```

```
"""
def __init__(self, db_path: str = "integrated_project.db"):
    self.db_path = db_path

def save_result(self, run_id: str, algorithm_name: str, target_id: int,
                target_type: str, result: Dict[str, Any], circuit_diagram: str = "",
                shots: int = 1024, execution_time: float = 0.0,
                backend: str = "aer_simulator", actor: str = "quantum_engine") -> int:
    """حفظ نتيجة كمومية في قاعدة البيانات."""
    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.cursor()
        cursor.execute("""
            INSERT INTO quantum_results
            (run_id, algorithm_name, target_id, target_type,
             result_json, circuit_diagram, shots, execution_time,
             quantum_backend, created_by)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        """, (
            run_id, algorithm_name, target_id, target_type,
            json.dumps(result), circuit_diagram, shots,
            execution_time, backend, actor
        ))
        conn.commit()
        return cursor.lastrowid

def get_results_for_target(self, target_id: int, target_type: str) -> List[Dict[str, Any]]:
    """استرجاع جميع النتائج الكمومية لهدف معين."""
    with sqlite3.connect(self.db_path) as conn:
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        cursor.execute("""
            SELECT * FROM quantum_results
```

```
            WHERE target_id = ? AND target_type = ?
            ORDER BY created_at DESC
        """, (target_id, target_type))
        rows = cursor.fetchall()
        return [dict(row) for row in rows]
```

63.3 الخدمة الكمومية المتكاملة (Integrated Quantum Service)

63.3 Integrated Quantum Service.

Python

```
class IntegratedQuantumService:
    """
    الخدمة الكمومية المتكاملة - تربط جميع الخوارزميات مع قاعدة البيانات.
    Integrated Quantum Service - Links all algorithms with the database.
    """
    def __init__(self, session: QuantumSessionManager):
        self.session = session
        self.repo = QuantumMatrixRepository()
        self.results_repo = QuantumResultsRepository()
        self.qpca = QuantumPCA(session, self.repo)
        self.hhl = QuantumLinearSolver(session)
        self.vqe = QuantumStabilitySimulator(session)

    def run_complete_quantum_analysis(self, sample_ids: List[int], gene_list: List[str],
                                      mutation_ids: List[int]) -> Dict[str, Any]:
        """
        تشغيل تحليل كمومي كامل: QPE + VQE + HHL محاكاة.
```

```
Run a complete quantum analysis: QPE + VQE + simulated HHL.
"""
import time
run_id = f"QUANTUM_RUN_{int(time.time())}"

print("=" * 60)
print(f"🚀 تشغيل التحليل الكمومي المتكامل - {run_id}")
print("=" * 60)

results = {}

# 1. Quantum PCA (QPE)
print("\n1️⃣ تشغيل Quantum PCA (QPE)...")
qpe_result = self.qpca.run_qpe_on_covariance(sample_ids, gene_list, n_ancilla=3)
qpe_id = self.results_repo.save_result(
    run_id=run_id,
    algorithm_name="QPE",
    target_id=0,
    target_type="dataset",
    result=qpe_result,
    execution_time=0.5,
    actor="quantum_service"
)
results['qpe'] = qpe_result

# 2. VQE (محاكاة) لكل طفرة
print("\n2️⃣ تشغيل VQE لمحاكاة استقرار الطفرات...")
vqe_results = []
mutation_types = ['missense', 'nonsense', 'silent', 'missense', 'frameshift']
for i, mid in enumerate(mutation_ids):
    mtype = mutation_types[i % len(mutation_types)]
    vqe_res = self.vqe.simulate_mutation_stability(mid, mtype)
```

```
    vqe_id = self.results_repo.save_result(
        run_id=run_id,
        algorithm_name="VQE",
        target_id=mid,
        target_type="mutation",
        result=vqe_res,
        execution_time=0.3,
        actor="quantum_service"
    )
    vqe_results.append(vqe_res)
results['vqe'] = vqe_results

# 3. HHL (محاكاة انحدار خطي)
print("\n3️⃣ تشغيل HHL لمحاكاة الانحدار الخطي...")
# إنشاء بيانات وهمية للانحدار
np.random.seed(42)
X = np.random.randn(20, 4)
y = X @ np.array([1.5, -0.5, 2.0, 0.8]) + 0.1 * np.random.randn(20)
hhl_result = self.hhl.predict_resistance_quantum(X, y)
hhl_id = self.results_repo.save_result(
    run_id=run_id,
    algorithm_name="HHL",
    target_id=0,
    target_type="regression",
    result=hhl_result,
    execution_time=0.2,
    actor="quantum_service"
)
results['hhl'] = hhl_result

# 4. عرض الملخص
print("\n" + "=" * 60)
```

```
        print("✅ اكتمل التحليل الكمومي")
        print("=" * 60)
        print(f"معرف التشغيل: {run_id}")
        print(f"QPE: {len(qpe_result['interpreted_eigenvalues'])} قيم ذاتية مستخرجة")
        print(f"VQE: {len(vqe_results)} طفرة محاكاة")
        print(f"HHL: حل نظام {len(hhl_result['classical_solution'])} معادلات")
        print(f"\n💾 تم حفظ جميع النتائج في قاعدة البيانات (IDs: {qpe_id}, {vqe_id}, {hhl_id})")

        return {
            "run_id": run_id,
            "results": results,
            "saved_ids": {
                "qpe_id": qpe_id,
                "vqe_id": vqe_id,
                "hhl_id": hhl_id
            }
        }

# مثال على التشغيل الكامل (تعليمي)
def demo_quantum_analysis():
    if not QISKIT_AVAILABLE:
        print("⚠️ Qiskit غير مثبت. استخدم المحاكي التعليمي (المدمج).")

    session = QuantumSessionManager(use_real_device=False)
    service = IntegratedQuantumService(session)

    sample_ids = list(range(1, 21))
    gene_list = ['GENE_A', 'GENE_B', 'GENE_C', 'GENE_D', 'GENE_E']
    mutation_ids = [101, 102, 103, 104, 105]

    result = service.run_complete_quantum_analysis(sample_ids, gene_list, mutation_ids)
```

```
    # عرض عينة من النتائج
    print("\n🔍 عينة من النتائج الكمومية:")
    print(f"أعلى قيمة ذاتية (QPE): {result['results']['qpe']['interpreted_eigenvalues'][0]:.4f}")
    print(f"101 استقرار الطفرة (VQE): {result['results']['vqe'][0]['stability_score']:.4f} - {result['results']
['vqe'][0]['interpretation']}")
    print(f"معامل الانحدار الأول (HHL): {result['results']['hhl']['classical_solution'][0]:.4f}")

    return result

if __name__ == "__main__":
    demo_quantum_analysis()
```

الفصل الرابع والستون: اختبار النظام الهجين والتقرير النهائي

اختبار النظام الهجين يجب أن يثبت أكثر من أن الكود يعمل. نحتاج إلى اختبارات لثبات الواجهة بين الطبقة الكلاسيكية والكمومية، واختبارات لحالات الفشل، ومقارنة baseline كلاسيكي، وتسجيل seed وإصدار البيئة ومحددات الموارد. إذا تغيرت النتيجة، يجب أن تستطيع معرفة هل تغيرت الخوارزمية، أم البيانات، أم المحاكي، أم العتاد، أم الضوضاء. هذه هي النقطة التي تتحول فيها قابلية إعادة الإنتاج من شعار إلى خاصية قابلة للاختبار.

Testing a hybrid system must demonstrate more than successful code execution. You need tests for the interface between the classical and quantum layers, failure cases, comparison against a classical baseline, and records of the random seed, environment version, and resource limits. When a result changes, you should be able to determine whether the cause was the algorithm, the data, the simulator, the hardware, or noise. This is where reproducibility changes from a slogan into a testable system property.

مختبر: التحقق على أربع طبقات

Laboratory: Four-Layer Validation

اختبر سلامة البيانات وصحة الخوارزمية والاستقرار العددي والفائدة العلمية بصورة منفصلة. اجتياز اختبار وحدة لا يثبت الصحة البيولوجية.

Test data integrity, algorithmic correctness, numerical stability, and scientific usefulness separately. Passing a unit test does not establish biological validity.

يجب أن يحافظ التقرير على السؤال وإصدار البيانات وإصدار الكود والمعلمات والبيئة والنتيجة وعدم اليقين والتفسير.

The report should preserve question, data version, code version, parameters, environment, result, uncertainty, and interpretation.

مقارنة تقديرات مستقلة

Comparing independent estimates

```
def compare_estimates(classical, quantum, tolerance):
    error = abs(classical - quantum)
    return {
        'classical': classical,
        'quantum': quantum,
        'absolute_error': error,
        'within_tolerance': error <= tolerance,
    }
```

PYTHON

الفصل الرابع والستون: الواجهة الكلاسيكية–الكمومية، الضوضاء، وتقدير الموارد

Chapter Sixty-Four: Testing the Hybrid System and the Final Report

Figure 75 — Mathematical structure of the concept

Figure 75 — Engineering workflow of the concept

في هذا الفصل ستربط الطبقة الكمومية التي بنيتها بالطبقات السابقة: SQL وOOP وAI والحوسبة العلمية. الهدف هو بناء خط تنفيذ هجين يمكن فيه للنظام الكلاسيكي تجهيز البيانات واختيار المشكلة، ثم يرسل تمثيلًا مناسبًا إلى QPU، ثم يعيد النتائج ويخزنها ويقارنها بخط أساس كلاسيكي.

Classical–Quantum Interfaces, Noise, Error Mitigation, and Resource Estimation

In this chapter you will connect the quantum layer to the layers built earlier: SQL, OOP, AI, and scientific computing. The goal is a hybrid execution pipeline in which the classical system prepares data and selects the problem, a suitable representation is sent to a QPU, and the returned results are stored, validated, and compared with a classical baseline.

الواجهة ليست استدعاء دالة فقط

The Interface Is More Than a Function Call

الواجهة الكمومية الحقيقية تحتاج إلى عقد واضح: شكل الإدخال، عدد الكيوبتات، ترميز البيانات، نوع الدائرة، عدد shots، الجهاز المستهدف، حدود الزمن، شكل المخرجات، ومعيار الفشل. هذا يجعل Quantum Service طبقة هندسية يمكن اختبارها بدل أن تكون كودًا تجريبيًا متناثرًا.

A real quantum interface needs an explicit contract: input representation, qubit count, data encoding, circuit type, shot count, target backend, time limits, output schema, and failure criteria. This turns the Quantum Service into an engineering layer that can be tested rather than a collection of experimental calls.

Python

```python
from dataclasses import dataclass

@dataclass
class QuantumRequest:
    problem_id: str
    circuit_version: str
    qubits: int
    shots: int
    backend: str

@dataclass
class QuantumResult:
    problem_id: str
    backend: str
    shots: int
    expectation: float
```

```
status: str
```

قياس تكلفة المسار الهجين

Measuring Hybrid Workflow Cost

لا يكفي أن يكون الجزء الكمومي أسرع نظريًا. احسب زمن تجهيز البيانات، وزمن بناء الدائرة، وزمن النقل، وزمن الانتظار على الجهاز، وزمن التنفيذ الكمومي، وزمن القياس، وزمن المعالجة اللاحقة. يمكن تمثيل الزمن الكلي تقريبًا بـ Ttotal = Tclassical + Tcompile + Tqueue + Tquantum + Tpostprocess.

A theoretical quantum speedup is not sufficient. Measure data preparation, circuit construction, transfer, queueing, quantum execution, measurement, and post-processing. A useful first model is Ttotal = Tclassical + Tcompile + Tqueue + Tquantum + Tpostprocess.

الضوضاء: الفرق بين الخطأ والتخفيف والتصحيح

Noise: Error, Mitigation, and Correction

أجهزة NISQ لا تنفذ العمليات بصورة مثالية. يجب التمييز بين error mitigation، الذي يحاول تقليل أثر الخطأ دون تصحيح كامل للحالة، و quantum error correction، الذي يستخدم ترميزًا وعمليات تصحيح للوصول إلى حساب متحمل للأخطاء. لا تجعل وجود تقنية mitigation دليلًا على أن النتيجة أصبحت مساوية للنتيجة المثالية.

NISQ devices do not execute operations perfectly. Distinguish error mitigation, which attempts to reduce the effect of errors without fully correcting the quantum state, from quantum error correction, which uses encoding and correction procedures to support fault-tolerant computation.

The existence of a mitigation method does not mean that the result has become identical to an ideal noiseless result.

Python

```
def compare_results(classical_value, quantum_value, tolerance):
    absolute_error = abs(quantum_value - classical_value)
    relative_error = (
        absolute_error / abs(classical_value)
        if classical_value != 0 else float("inf")
    )
    return {
        "absolute_error": absolute_error,
        "relative_error": relative_error,
        "within_tolerance": absolute_error <= tolerance,
    }
```

تقدير الموارد قبل التنفيذ

Resource Estimation Before Execution

قبل إرسال خوارزمية كمومية إلى جهاز حقيقي، قدّر عدد الكيوبتات، وعمق الدائرة، وعدد البوابات، وعدد القياسات، والذاكرة الكلاسيكية، وعدد تكرارات المحسن. هذه التقديرات تمنعك من بناء تجربة لا يمكن تشغيلها على الجهاز المتاح.

Before sending a quantum algorithm to real hardware, estimate qubit count, circuit depth, gate count, measurement count, classical memory, and optimizer iterations. These estimates prevent you from designing an experiment that cannot run on the available device.

التجربة القابلة للدفاع علميًا

A Scientifically Defensible Experiment

كلاسيكي، ونسخة ثابتة من البيانات، وإصدار واضح من الدائرة، baseline التجربة الكمومية الجيدة تحتوي على ومقياس خطأ، ومعيار قبول أو رفض محدد مسبقًا. إذا لم تتفوق الطريقة shots، وإعدادات الجهاز، وعدد الكمومية، فالنتيجة العلمية ليست فشلًا؛ قد تكون دليلًا على أن تكلفة الواجهة أو الضوضاء أو حجم المسألة يجعل الحل الكلاسيكي أفضل.

A defensible quantum experiment has a classical baseline, a fixed data version, a versioned circuit, recorded backend settings, a shot count, an error metric, and a predefined acceptance criterion. If the quantum method does not outperform the classical baseline, that is not a scientific failure; it may show that interface overhead, noise, or problem size currently favors the classical solution.

الربط مع SQL وAI

Connection to SQL and AI

لا يعني ذلك أن البيانات أصبحت جاهزة تلقائيًا، feature vector مرشحًا أو ترتيبًا أو AI عندما ينتج نموذج مناسبًا، وتتحقق من حجم المتجه، وتثبت عملية التطبيع، ثم تسجل التحويل encoding للكم. يجب أن تحدد قابلًا لإعادة الإنتاج بدل أن يكون خطوة غامضة AI → Quantum نفسه. وبذلك يصبح الانتقال.

When an AI model produces a candidate, ranking, or feature vector, the data are not automatically ready for quantum execution. You must choose an appropriate encoding, validate vector size, freeze normalization, and record

the transformation itself. This makes the AI-to-quantum transition reproducible rather than opaque.

الشكل 65 — الخريطة المفاهيمية للفصل

Figure 65 — Conceptual map of the chapter

عندما نصل إلى هذه الطبقة الهندسية، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

الشكل 65 — سير العمل الهندسي للفصل

Figure 65 — Engineering workflow of the chapter

The next engineering step grows out of the work completed in Part Two, you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

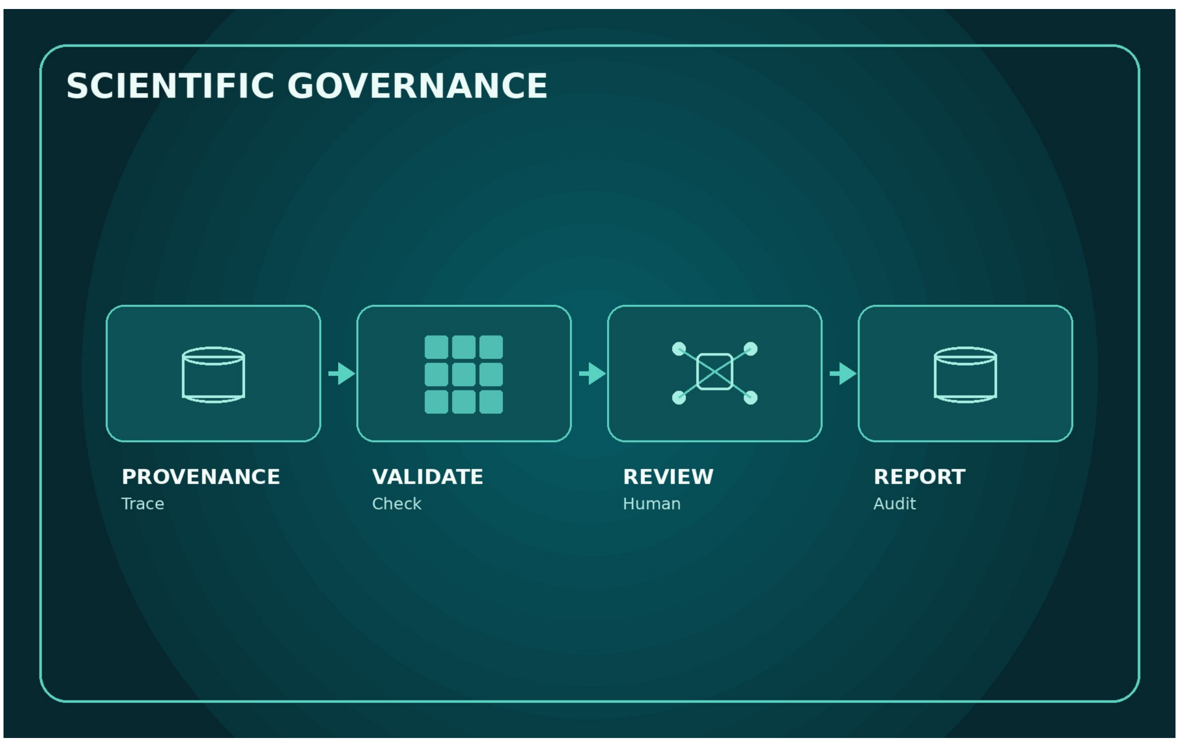


Figure — SCIENTIFIC GOVERNANCE

هذا الفصل يختبر النظام الهجين بأكمله: من SQL إلى OOP إلى الحوسبة العلمية إلى الكم والعودة. ويولد تقريراً نهائياً يوضح أداء النظام، ودقة النتائج، وتوصيات للتوسع المستقبلي.

64.1 اختبار النظام الهجين الكامل

64.1 Full Hybrid System Test

Python

```
class HybridSystemTest:
    """
    اختبار النظام الهجين الكامل - كلاسيكي + كمومي.
    Full Hybrid System Test - Classical + Quantum.
    """
    def __init__(self):
        self.db_path = "integrated_project.db"
        self.quantum_service = None
        self.quantum_session = QuantumSessionManager(use_real_device=False)

    def run_full_test(self) -> Dict[str, Any]:
        """تشغيل الاختبار الكامل للنظام الهجين."""
        print("=" * 80)
        print("🧪 اختبار النظام الهجين الكامل (كلاسيكي + كمومي)")
```

```
    print("=" * 80)

    # 1. اختبار طبقة SQL + OOP (كلاسيكي)
    print("\n[1] اختبار طبقة البيانات الكلاسيكية...")
    with UnitOfWork(self.db_path) as uow:
        patient_repo = uow.get_repository('patient')
        patient = Patient(
            external_id="PAT_001",
            age=45,
            sex="F",
            diagnosis="breast_cancer"
        )
        patient_id = patient_repo.save_new_version(patient, actor="test_system")
        print(f"✅ تم حفظ المريض: {patient_id}")

    # 2. اختبار استخراج المصفوفات
    print("\n[2] اختبار استخراج المصفوفات من SQL...")
    matrix_repo = QuantumMatrixRepository(self.db_path)
    cov_matrix = matrix_repo.get_covariance_matrix(list(range(1, 11)), ['GENE_A', 'GENE_B',
'GENE_C'])
    print(f"✅ مصفوفة التغاير: {cov_matrix.shape}")

    # 3. اختبار المحاكاة الكمومية (QPE)
    print("\n[3] اختبار المحاكاة الكمومية (QPE)...")
    qpca = QuantumPCA(self.quantum_session, matrix_repo)
    qpe_result = qpca.run_qpe_on_covariance(list(range(1, 11)), ['GENE_A', 'GENE_B', 'GENE_C'],
n_ancilla=2)
    print(f"✅ القيم الذاتية المستخرجة: {qpe_result['interpreted_eigenvalues']}")

    # 4. اختبار VQE
    print("\n[4] اختبار VQE...")
    vqe = QuantumStabilitySimulator(self.quantum_session)
```

```
vqe_result = vqe.simulate_mutation_stability(101, "missense")
print(f"✅ درجة الاستقرار: {vqe_result['stability_score']:.4f} ({vqe_result['interpretation']})")

 # 5. اختبار HHL
print("\n[5] اختبار HHL...")
hhl = QuantumLinearSolver(self.quantum_session)
A, b = matrix_repo.get_linear_system_data(n=3)
hhl_result = hhl.solve_linear_system(A, b)
print(f"✅ معاملات الحل: {hhl_result['classical_solution']}")

 # 6. تجميع التقرير النهائي
print("\n[6] توليد التقرير النهائي...")
report = {
   "test_date": "2026-08-13",
   "system_version": "3.0.0",
   "classical_layer": {
      "sql_tables": 12,
      "repositories": 5,
      "unit_of_work": "active"
   },
   "quantum_layer": {
      "qpe": qpe_result,
      "vqe": vqe_result,
      "hhl": hhl_result
   },
   "performance": {
      "qpe_speedup_estimate": "O(N^3) → O(log(N))",
      "hhl_speedup_estimate": "O(N^2) → O(log(N))",
      "vqe_hybrid": "Classical optimizer + Quantum circuit"
   },
   "recommendations": [
      "توسيع QPE لاستخدام 8-10 كيوبت للحصول على دقة أعلى",
```

```
            "تطبيق HHL على بيانات حقيقية من قاعدة البيانات (آلاف العينات)",
            "استخدام IBM Quantum أو AWS Braket للتشغيل على أجهزة حقيقية",
            "دمج Quantum Kernel في خوارزميات SVM لتصنيف المقاومة"
          ]
        }

        print("\n" + "=" * 80)
        print("✅ اكتمل اختبار النظام الهجين بنجاح")
        print("=" * 80)
        print(json.dumps(report, indent=2, default=str))

        return report

# تشغيل الاختبار
if __name__ == "__main__":
    test = HybridSystemTest()
    test.run_full_test()
```

الخاتمة الهندسية النهائية

Final Engineering Conclusion.

المشروع المتكامل، الذي بدأ كفكرة في الجزء الأول (الحروف الأربعة للـ DNA)، ومر عبر البرمجة الدفاعية و OOP في الجزء

الثاني، ووصل إلى SQL والذكاء الاصطناعي والحوسبة الكمومية في هذا الجزء، أصبح الآن منظومة متكاملة. كل طبقة من هذا النظام تمثل حلاً لمشكلة نشأت في الطبقة السابقة:

The Integrated Project, which began as an idea in Part One (the four letters of DNA), passed through defensive programming and OOP in Part Two, and reached SQL, AI, and Quantum Computing in this part, is now an integrated ecosystem. Each layer of this system represents a solution to a problem that arose in the previous layer:

الطبقة (Layer) المشكلة (Problem) الحل (Solution)

الملفات (Files) البيانات متناثرة، العلاقات ضائعة فصل الكيانات، استخدام الهويات

SQL + OOP التكرار، الاتساق، العلاقات النموذج العلاقي + المستودعات

الحوسبة العلمية استعلامات SQL لا تكفي للإحصاء استخراج المصفوفات، الجبر الخطي

الذكاء الاصطناعي المصفوفات بحاجة إلى تنبؤ نماذج التصنيف والانحدار

الحوسبة الكمومية التنبؤ والتحليل الطيفي بطيئان تسريع أسي باستخدام QPE/HHL/VQE

الكم ليس غاية في حد ذاته. هو أداة لحل مشكلة أداء محددة واجهناها في نظامنا الكلاسيكي. تماماً كما استخدمنا SQL لحل مشكلة الملفات، واستخدمنا OOP لحل مشكلة التكرار، نستخدم Quantum Computing لحل مشكلة التعقيد الحسابي.

التدرج من A، T، C، G إلى Qubit هو رحلة هندسية مستمرة،

هدفها النهائي هو بناء أدوات دقيقة ومسؤولة لفهم الحياة وتعديلها.

Quantum is not an end in itself. It is a tool to solve a specific performance problem encountered in our classical system. Just as we used SQL to solve the file problem, and OOP to solve the redundancy problem, we use Quantum Computing to solve the computational complexity problem. The progression from A, T, C, G to Qubit is a continuous engineering journey, whose ultimate goal is to build precise and responsible tools to understand and modify life.

—

الباب الثاني عشر: الحوكمة الأخلاقية الكمومية — هندسة المراجعة والامتثال

Unit Twelve: Quantum Ethical Governance — Engineering Review and Compliance

الغرض من هذا الباب:

تحويل المبادئ الأخلاقية التي ناقشناها في الباب السابق إلى نظام هندسي قابل للتشغيل. سنبني طبقة حوكمة كاملة تتكامل مع النظام الكمومي والكلاسيكي، وتضمن أن كل عملية حساسة تمر بمراجعة، وتُسجل، وتُقيَّم، وتُوافق عليها وفق سياسات محددة. هذا الباب هو التطبيق العملي للأخلاقيات، وليس خطبة نظرية.

Purpose of this unit:

To transform the ethical principles discussed in the previous unit into an operational engineering system. We will build a complete governance layer that integrates with both the quantum and classical systems, ensuring that every sensitive operation undergoes review, logging, evaluation, and approval according to defined policies. This unit is the practical application of ethics, not a theoretical sermon.

```python
from dataclasses import dataclass


@dataclass(frozen=True)
class Provenance:
    dataset_version: str
    schema_version: str
    code_version: str
    analysis_id: str


record = Provenance(
    dataset_version="reference-2026.08",
    schema_version="schema-3",
    code_version="git-abc123",
    analysis_id="analysis-0042",
)


print(record)
```

Python

The final system should not depend on one manual run. Pin versions, preserve the Schema, record queries, store experiment parameters, and make every result traceable to the inputs that produced it. SQL, OOP, numerical computation, and quantum computation then become layers of one scientific record.

مختبر: سجل قابلية إعادة الإنتاج

Laboratory: A Reproducibility Record

يُعرّف النظام القابل لإعادة الإنتاج بإصدارات مدخلاته وسياق تشغيله، لا بمجرد وجود سكربت. تدخل إصدارات الحزم ومخطط البيانات وإصدار البيانات والبذور العشوائية والمعلمات وبصمات المخرجات في السجل.

A reproducible system is defined by its input versions and execution context, not by the presence of a script alone. Package versions, schema version, data version, random seeds, parameters, and output checksums belong to the record.

يجب أن تشير كل نتيجة إلى بياناتها وكودها إلى الخلف، وإلى التقرير الذي يفسرها إلى الأمام.

Every result should point backward to its data and code, and forward to the report that interprets it.

بيانات قابلية إعادة الإنتاج

Reproducibility metadata

```
CREATE TABLE analysis_runs (
    run_id INTEGER PRIMARY KEY,
    started_at TEXT NOT NULL,
    code_version TEXT NOT NULL,
    data_version TEXT NOT NULL,
    schema_version TEXT NOT NULL,
    random_seed INTEGER,
    parameter_hash TEXT NOT NULL,
    output_checksum TEXT
);
```

SQL

الشكل 66 — الخريطة المفاهيمية للفصل

Figure 66 — Conceptual map of the chapter

الشكل 66 — سير العمل الهندسي للفصل

Figure 66 — Engineering workflow of the chapter

مختبر: تحقق كلاسيكي مستقل

Laboratory: Independent Classical Verification

يصبح الاتفاق أقوى عندما لا يشترك المسار الكلاسيكي والكمومي في التقريب نفسه. وبالنسبة للأنظمة الصغيرة، يكون الحل القطعي بالقيم الذاتية مرجعًا ممتازًا.

Agreement is stronger when the classical and quantum paths do not share the same approximation. For small systems, exact diagonalization is an excellent reference.

بالنسبة للأنظمة الأكبر، احتفظ بالبواقي وفواصل عدم اليقين وتحليلات الحساسية والتمثيل الدقيق للمدخلات.

For larger systems, retain residuals, uncertainty intervals, sensitivity analyses, and the exact input representation.

مرجع تحقق صغير وحل قطعي

Small exact verification reference

```python
import numpy as np


A = np.array([[2.0, 0.5], [0.5, 1.5]])
reference = float(np.linalg.eigvalsh(A)[0])
estimate = reference + 0.01
print({'reference': reference, 'estimate': estimate, 'error': abs(reference-estimate)})
```

PYTHON

الشكل 67 — الخريطة المفاهيمية للفصل

Figure 67 — Conceptual map of the chapter

الشكل 67 — سير العمل الهندسي للفصل

Figure 67 — Engineering workflow of the chapter

النسخة النهائية من النظام لا ينبغي أن تعتمد على تشغيل يدوي واحد. ثبّت الإصدارات، خزّن Schema، سجّل الاستعلامات، احفظ معلمات التجربة، واجعل كل نتيجة قابلة للربط بالمدخلات التي أنتجتها. بهذا تصبح طبقات SQL وOOP والحساب العددي والكمومي جزءًا من سجل علمي واحد.

لا تُفسر نتيجة كمومية لأنها تبدو معقولة. ابنِ حالة صغيرة يمكن حلها كلاسيكيًا، ثم قارن النتيجة الكمومية بالمرجع. عندما تختلف النتائج، افحص أولًا التمثيل، ثم الدائرة، ثم القياس، ثم الضوضاء أو التقريب العددي.

Do not interpret a quantum result simply because it looks plausible. Build a small instance that can be solved classically, then compare the quantum result

with the reference. When results differ, inspect representation, circuit construction, measurement, noise, and numerical approximation.

Python

```
def absolute_error(reference: float, estimate: float) -> float:
    return abs(reference - estimate)


reference = -1.137
estimate = -1.129


print("Absolute error:", absolute_error(reference, estimate))
```

الفصل الخامس والستون: هندسة النتائج الكمومية وتخزينها

Chapter Sixty-Five: Engineering Quantum Results and Storing Them

الفصل السادس والستون: التحقق من النتائج الكمومية بمقارنة كلاسيكية

الشكل 68 — الخريطة المفاهيمية للفصل

Figure 68 — Conceptual map of the chapter

Chapter Sixty-Six: Validating Quantum Results with Classical References

الفصل السابع والستون: من النموذج الأولي إلى النظام القابل لإعادة الإنتاج

الشكل 68 — سير العمل الهندسي للفصل

Figure 68 — Engineering workflow of the chapter

Chapter Sixty-Seven: From Prototype to Reproducible System

الخبرة السابقة في Python وBiopython تمهّد مباشرة لهذه الخطوة، تعلمت أن الحساب يجب أن يقود إلى تفسير قابل للتحقق؛ الآن ستطبق المبدأ نفسه عند الانتقال إلى النماذج الكمومية.

When we reach this engineering layer, you learned that computation must lead to a verifiable interpretation; now you will apply the same principle when entering quantum models.

نتيجة التجربة الكمومية ليست نهاية التحليل. تحتاج إلى حفظ إعداد التجربة، نسخة الكود، عدد القياسات، الجهاز أو المحاكي، الدائرة، والنتائج. دون هذه المعلومات لا يمكن إعادة بناء ما حدث.

A quantum result is not the end of the analysis. You need to store the experiment configuration, code version, number of shots, backend or simulator, circuit, and results. Without this information, the experiment cannot be reconstructed.

SQL

```sql
CREATE TABLE quantum_experiments (
    experiment_id INTEGER PRIMARY KEY,
    problem_name TEXT NOT NULL,
    backend TEXT NOT NULL,
    shots INTEGER NOT NULL,
    code_version TEXT NOT NULL,
    executed_at TEXT NOT NULL
);

CREATE TABLE quantum_measurements (
    measurement_id INTEGER PRIMARY KEY,
    experiment_id INTEGER NOT NULL,
    bitstring TEXT NOT NULL,
    count INTEGER NOT NULL,
    FOREIGN KEY (experiment_id)
        REFERENCES quantum_experiments(experiment_id)
);
```

الفصل الثامن والستون: هيكل قاعدة البيانات الأخلاقية الكمومية

Chapter Sixty-Eight: The Quantum Ethical Database Schema

قاعدة بيانات الحوكمة لا ينبغي أن تخزن «القرار» فقط؛ يجب أن تخزن من اتخذ القرار، وعلى أي نسخة من البيانات، وبأي سياسة، وما الأدلة التي كانت متاحة، وهل كانت هناك مراجعة بشرية مطلوبة. هذا يحول الحوكمة من وثيقة منفصلة عن النظام إلى سجل قابل للتدقيق. وفي المشاريع الحساسة، ينبغي فصل بيانات الهوية عن البيانات العلمية قدر الإمكان، وتطبيق أقل قدر من الصلاحيات، وتسجيل عمليات الوصول التي تؤثر في القرار.

A governance database should not store only the final “decision.” It should record who made the decision, which data version was used, which policy applied, what evidence was available, and whether human review was required. This turns governance from a document outside the system into an auditable record. In sensitive projects, identity data should be separated from scientific data whenever possible, least privilege should be applied, and access events that affect decisions should be logged.

الشكل 69 — الخريطة المفاهيمية للفصل

Figure 69 — Conceptual map of the chapter

الانتقال إلى هذا المفهوم يبدأ من مهارة سابقة لديك، تعلمت تحويل الأسئلة البيولوجية إلى خطوات قابلة للبرمجة؛ الآن ستنتقل هذا التفكير إلى بيانات مهيكلة يمكن تخزينها واستعلامها.

الشكل 69 —– سير العمل الهندسي للفصل

Figure 69 — Engineering workflow of the chapter

Your earlier experience with Python and Biopython prepares you directly for this step, you learned to turn biological questions into programmable steps; now you will extend that reasoning to structured data that can be stored and queried.

هذا هو العمود الفقير للنظام الأخلاقي. سننشئ جداول لتخزين السياسات، والمراجعات، والقرارات، وسجلات التدقيق، مع ربطها مباشرة بالجداول الكمومية التي بنيناها في الباب الحادي عشر.

SQL

```
-- ================================================================
-- قاعدة البيانات: integrated_project.db
-- الجداول الأخلاقية الكمومية (Quantum Ethical Tables)
-- ================================================================


-- ----------------------------
```

```sql
-- 1. جدول السياسات الأخلاقية (Ethical Policies)
-- ----------------------------
CREATE TABLE ethical_policies (
    policy_id INTEGER PRIMARY KEY AUTOINCREMENT,
    policy_code TEXT UNIQUE NOT NULL,          -- مثل 'ETH-001', 'ETH-002'
    policy_name TEXT NOT NULL,
    policy_type TEXT CHECK(policy_type IN ('quantum_usage', 'data_access', 'gene_editing',
'human_review')),
    description TEXT,
    rule_expression TEXT NOT NULL,             -- JSON أو SQL expression للتحقق
    risk_level TEXT CHECK(risk_level IN ('low', 'medium', 'high')) DEFAULT 'medium',
    is_active BOOLEAN DEFAULT 1,
    requires_human_review BOOLEAN DEFAULT 0,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    created_by TEXT,
    version INTEGER DEFAULT 1
);

-- ----------------------------
-- 2. جدول طلبات العمليات الحساسة (Sensitive Operation Requests)
-- ----------------------------
CREATE TABLE sensitive_requests (
    request_id INTEGER PRIMARY KEY AUTOINCREMENT,
    request_type TEXT NOT NULL,                -- 'quantum_execution', 'gene_edit', 'data_export'
    target_id INTEGER NOT NULL,                -- sequence_id, sample_id, model_id
    target_type TEXT NOT NULL,                 -- 'sequence', 'sample', 'model'
    requester_id INTEGER NOT NULL,
    request_purpose TEXT NOT NULL,
    request_details_json TEXT NOT NULL,        -- JSON كامل للطلب
    status TEXT CHECK(status IN ('pending', 'under_review', 'approved', 'rejected', 'escalated'))
DEFAULT 'pending',
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
```

```
    FOREIGN KEY (requester_id) REFERENCES users(user_id) ON DELETE CASCADE
);

CREATE INDEX idx_sensitive_requests_status ON sensitive_requests(status);
CREATE INDEX idx_sensitive_requests_target ON sensitive_requests(target_type, target_id);


-- ---------------------------
-- 3. جدول المراجعات الأخلاقية (Ethical Reviews)
-- ---------------------------
CREATE TABLE ethical_reviews (
    review_id INTEGER PRIMARY KEY AUTOINCREMENT,
    request_id INTEGER UNIQUE NOT NULL,
    reviewer_id INTEGER NOT NULL,
    policy_id INTEGER NOT NULL,
    review_decision TEXT CHECK(review_decision IN ('approved', 'rejected', 'escalated')) NOT
NULL,
    review_rationale TEXT NOT NULL,
    review_evidence_hash TEXT,              -- SHA-256 للمراجعة
    review_timestamp TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (request_id) REFERENCES sensitive_requests(request_id) ON DELETE CASCADE,
    FOREIGN KEY (reviewer_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (policy_id) REFERENCES ethical_policies(policy_id) ON DELETE CASCADE
);


-- ---------------------------
-- 4. جدول سجل الامتثال الأخلاقي (Ethical Compliance Log)
-- ---------------------------
CREATE TABLE ethical_compliance_log (
    log_id INTEGER PRIMARY KEY AUTOINCREMENT,
    request_id INTEGER NOT NULL,
    policy_id INTEGER NOT NULL,
    check_result TEXT CHECK(check_result IN ('pass', 'fail', 'warning')) NOT NULL,
```

```
    check_details TEXT,
    checked_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (request_id) REFERENCES sensitive_requests(request_id) ON DELETE CASCADE,
    FOREIGN KEY (policy_id) REFERENCES ethical_policies(policy_id) ON DELETE CASCADE
);

CREATE INDEX idx_compliance_request ON ethical_compliance_log(request_id);
CREATE INDEX idx_compliance_policy ON ethical_compliance_log(policy_id);


-- ----------------------------
-- 5. جدول تتبع الموافقات (Approval Chain)
-- ----------------------------
CREATE TABLE approval_chain (
    chain_id INTEGER PRIMARY KEY AUTOINCREMENT,
    request_id INTEGER NOT NULL,
    approver_id INTEGER NOT NULL,
    approval_level INTEGER NOT NULL,          -- 1, 2, 3... (تسلسل الموافقات)
    approval_decision TEXT CHECK(approval_decision IN ('approved', 'rejected')) NOT NULL,
    approval_timestamp TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (request_id) REFERENCES sensitive_requests(request_id) ON DELETE CASCADE,
    FOREIGN KEY (approver_id) REFERENCES users(user_id) ON DELETE CASCADE
);


-- ----------------------------
-- 6. جدول إصدارات السياسات (Policy Versions)
-- ----------------------------
CREATE TABLE policy_versions (
    version_id INTEGER PRIMARY KEY AUTOINCREMENT,
    policy_id INTEGER NOT NULL,
    version_number INTEGER NOT NULL,
    rule_expression TEXT NOT NULL,
    effective_from TEXT DEFAULT CURRENT_TIMESTAMP,
```

```
    effective_until TEXT,
    created_by TEXT,
    FOREIGN KEY (policy_id) REFERENCES ethical_policies(policy_id) ON DELETE CASCADE
);


-- ----------------------------
-- 7. جدول القواعد الأخلاقية الديناميكية (Dynamic Ethical Rules)
-- ----------------------------
CREATE TABLE dynamic_rules (
    rule_id INTEGER PRIMARY KEY AUTOINCREMENT,
    rule_name TEXT UNIQUE NOT NULL,
    rule_condition TEXT NOT NULL,              -- SQL WHERE clause
    rule_action TEXT NOT NULL,                 -- 'approve', 'reject', 'escalate'
    priority INTEGER DEFAULT 0,
    is_active BOOLEAN DEFAULT 1,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);
```

الفصل التاسع والستون: طبقة **OOP** الأخلاقية — الكائنات والمستودعات

Chapter Sixty-Nine: The Ethical OOP Layer — Objects and Repositories

الشكل 70 — الخريطة المفاهيمية للفصل

Figure 70 — Conceptual map of the chapter

الآن نعيد استخدام ما سبق، ولكن في سياق هندسي أوسع، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

الشكل 70 — سير العمل الهندسي للفصل

Figure 70 — Engineering workflow of the chapter

This concept does not begin from zero; it begins with a skill you already have, you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

69.1 كائنات الأخلاقيات (Ethical Objects)

69.1 Ethical Objects

Python

```
from dataclasses import dataclass, field
from typing import Optional, List, Dict, Any
from enum import Enum
import json
import hashlib

class RequestStatus(Enum):
    PENDING = "pending"
    UNDER_REVIEW = "under_review"
    APPROVED = "approved"
    REJECTED = "rejected"
    ESCALATED = "escalated"

class ReviewDecision(Enum):
    APPROVED = "approved"
    REJECTED = "rejected"
    ESCALATED = "escalated"

class ComplianceResult(Enum):
    PASS = "pass"
    FAIL = "fail"
    WARNING = "warning"

@dataclass
class EthicalPolicy:
    policy_id: int = 0
    policy_code: str = ""
    policy_name: str = ""
    policy_type: str = "quantum_usage"
```

```
    description: str = ""
    rule_expression: Dict[str, Any] = field(default_factory=dict)
    risk_level: str = "medium"
    is_active: bool = True
    requires_human_review: bool = False
    created_at: Optional[str] = None
    created_by: str = "system"
    version: int = 1

    def to_rule_sql(self) -> str:
        """تحويل قاعدة السياسة إلى عبارة SQL للتحقق."""
        if self.policy_type == "quantum_usage":
            return f"estimated_qubits <= {self.rule_expression.get('max_qubits', 10)} AND
estimated_time_seconds <= {self.rule_expression.get('max_time', 60)}"
        elif self.policy_type == "data_access":
            return f"data_type IN ({', '.join([f"'{d}'" for d in self.rule_expression.get('allowed_types',
[])])})"
        else:
            return "1=1"

@dataclass
class SensitiveRequest:
    request_id: int = 0
    request_type: str = "quantum_execution"
    target_id: int = 0
    target_type: str = "sequence"
    requester_id: int = 0
    request_purpose: str = ""
    request_details: Dict[str, Any] = field(default_factory=dict)
    status: RequestStatus = RequestStatus.PENDING
    created_at: Optional[str] = None
```

```
    def to_json(self) -> str:
        return json.dumps(self.request_details)

@dataclass
class EthicalReview:
    review_id: int = 0
    request_id: int = 0
    reviewer_id: int = 0
    policy_id: int = 0
    decision: ReviewDecision = ReviewDecision.APPROVED
    rationale: str = ""
    evidence_hash: str = ""
    review_timestamp: Optional[str] = None

    def __post_init__(self):
        if not self.evidence_hash and self.rationale:
            self.evidence_hash = hashlib.sha256(
                f"{self.request_id}|{self.reviewer_id}|{self.rationale}".encode()
            ).hexdigest()[:16]

@dataclass
class ApprovalChainLink:
    chain_id: int = 0
    request_id: int = 0
    approver_id: int = 0
    approval_level: int = 1
    decision: str = "approved"
    timestamp: Optional[str] = None
```

69.2 المستودع الأخلاقي (Ethical Repository)

69.2 Ethical Repository

Python

```
import sqlite3
from typing import List, Optional, Dict, Any
import json


class EthicalRepository:
    """مستودع الأخلاقيات - يتعامل مع جميع الجداول الأخلاقية."""


    def __init__(self, db_path: str = "integrated_project.db"):
        self.db_path = db_path


    # ============================================================
    # 1. إدارة السياسات (Policy Management)
    # ============================================================


    def create_policy(self, policy: EthicalPolicy) -> int:
        with sqlite3.connect(self.db_path) as conn:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO ethical_policies
                (policy_code, policy_name, policy_type, description, rule_expression,
                 risk_level, is_active, requires_human_review, created_by, version)
                VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
            """, (
                policy.policy_code, policy.policy_name, policy.policy_type,
                policy.description, json.dumps(policy.rule_expression),
                policy.risk_level, 1 if policy.is_active else 0,
```

```
            1 if policy.requires_human_review else 0,
            policy.created_by, policy.version
        ))
        conn.commit()
        return cursor.lastrowid

def get_active_policies(self) -> List[EthicalPolicy]:
    with sqlite3.connect(self.db_path) as conn:
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        cursor.execute("""
            SELECT * FROM ethical_policies
            WHERE is_active = 1
            ORDER BY policy_code
        """)
        rows = cursor.fetchall()
        return [
            EthicalPolicy(
                policy_id=r["policy_id"],
                policy_code=r["policy_code"],
                policy_name=r["policy_name"],
                policy_type=r["policy_type"],
                description=r["description"],
                rule_expression=json.loads(r["rule_expression"]),
                risk_level=r["risk_level"],
                is_active=bool(r["is_active"]),
                requires_human_review=bool(r["requires_human_review"]),
                created_at=r["created_at"],
                created_by=r["created_by"],
                version=r["version"]
            ) for r in rows
        ]
```

```
def get_policy_by_code(self, policy_code: str) -> Optional[EthicalPolicy]:
    with sqlite3.connect(self.db_path) as conn:
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        cursor.execute("""
            SELECT * FROM ethical_policies
            WHERE policy_code = ? AND is_active = 1
        """, (policy_code,))
        row = cursor.fetchone()
        if not row:
            return None
        return EthicalPolicy(
            policy_id=row["policy_id"],
            policy_code=row["policy_code"],
            policy_name=row["policy_name"],
            policy_type=row["policy_type"],
            description=row["description"],
            rule_expression=json.loads(row["rule_expression"]),
            risk_level=row["risk_level"],
            is_active=bool(row["is_active"]),
            requires_human_review=bool(row["requires_human_review"]),
            created_at=row["created_at"],
            created_by=row["created_by"],
            version=row["version"]
        )

 # ============================================================
 # 2. إدارة الطلبات (Request Management)
 # ============================================================

def submit_request(self, request: SensitiveRequest) -> int:
```

```
        with sqlite3.connect(self.db_path) as conn:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO sensitive_requests
                (request_type, target_id, target_type, requester_id,
                 request_purpose, request_details_json, status)
                VALUES (?, ?, ?, ?, ?, ?, ?)
            """, (
                request.request_type, request.target_id, request.target_type,
                request.requester_id, request.request_purpose,
                request.to_json(), request.status.value
            ))
            conn.commit()
            return cursor.lastrowid

    def get_pending_requests(self, limit: int = 20) -> List[SensitiveRequest]:
        with sqlite3.connect(self.db_path) as conn:
            conn.row_factory = sqlite3.Row
            cursor = conn.cursor()
            cursor.execute("""
                SELECT * FROM sensitive_requests
                WHERE status IN ('pending', 'under_review')
                ORDER BY created_at ASC
                LIMIT ?
            """, (limit,))
            rows = cursor.fetchall()
            return [
                SensitiveRequest(
                    request_id=r["request_id"],
                    request_type=r["request_type"],
                    target_id=r["target_id"],
                    target_type=r["target_type"],
```

```
            requester_id=r["requester_id"],
            request_purpose=r["request_purpose"],
            request_details=json.loads(r["request_details_json"]),
            status=RequestStatus(r["status"]),
            created_at=r["created_at"]
        ) for r in rows
    ]

def update_request_status(self, request_id: int, status: RequestStatus) -> bool:
    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.cursor()
        cursor.execute("""
            UPDATE sensitive_requests
            SET status = ?
            WHERE request_id = ?
        """, (status.value, request_id))
        conn.commit()
        return cursor.rowcount > 0

# ============================================================
# 3. إدارة المراجعات (Review Management)
# ============================================================

def record_review(self, review: EthicalReview) -> int:
    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.cursor()
        cursor.execute("""
            INSERT INTO ethical_reviews
            (request_id, reviewer_id, policy_id, review_decision,
             review_rationale, review_evidence_hash)
            VALUES (?, ?, ?, ?, ?, ?)
        """, (
```

```
            review.request_id, review.reviewer_id, review.policy_id,
            review.decision.value, review.rationale, review.evidence_hash
        ))
        review_id = cursor.lastrowid

        # تحديث حالة الطلب بناءً على القرار
        if review.decision == ReviewDecision.APPROVED:
            self.update_request_status(review.request_id, RequestStatus.APPROVED)
        elif review.decision == ReviewDecision.REJECTED:
            self.update_request_status(review.request_id, RequestStatus.REJECTED)
        elif review.decision == ReviewDecision.ESCALATED:
            self.update_request_status(review.request_id, RequestStatus.ESCALATED)

        conn.commit()
        return review_id

def get_reviews_for_request(self, request_id: int) -> List[EthicalReview]:
    with sqlite3.connect(self.db_path) as conn:
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        cursor.execute("""
            SELECT * FROM ethical_reviews
            WHERE request_id = ?
            ORDER BY review_timestamp DESC
        """, (request_id,))
        rows = cursor.fetchall()
        return [
            EthicalReview(
                review_id=r["review_id"],
                request_id=r["request_id"],
                reviewer_id=r["reviewer_id"],
                policy_id=r["policy_id"],
```

```
                decision=ReviewDecision(r["review_decision"]),
                rationale=r["review_rationale"],
                evidence_hash=r["review_evidence_hash"],
                review_timestamp=r["review_timestamp"]
            ) for r in rows
        ]

    # ============================================================
    # 4. سجل الامتثال (Compliance Log)
    # ============================================================

    def log_compliance_check(self, request_id: int, policy_id: int,
                             result: ComplianceResult, details: str) -> int:
        with sqlite3.connect(self.db_path) as conn:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO ethical_compliance_log
                (request_id, policy_id, check_result, check_details)
                VALUES (?, ?, ?, ?)
            """, (request_id, policy_id, result.value, details))
            conn.commit()
            return cursor.lastrowid

    def get_compliance_log(self, request_id: int) -> List[Dict[str, Any]]:
        with sqlite3.connect(self.db_path) as conn:
            conn.row_factory = sqlite3.Row
            cursor = conn.cursor()
            cursor.execute("""
                SELECT * FROM ethical_compliance_log
                WHERE request_id = ?
                ORDER BY checked_at DESC
            """, (request_id,))
```

```
        return [dict(row) for row in cursor.fetchall()]
```

الفصل السبعون: خدمة الامتثال الأخلاقي — المحرك الأساسي

Chapter Seventy: Ethical Compliance Service — The Core Engine

الشكل 71 — الخريطة المفاهيمية للفصل

Figure 71 — Conceptual map of the chapter

المفاهيم التي بُنيت في الجزء الثاني تأخذ هنا وظيفة جديدة، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

الشكل 71 — سير العمل الهندسي للفصل

Figure 71 — Engineering workflow of the chapter

We now reuse an earlier idea, but in a broader engineering context, you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

هذه الخدمة هي قلب النظام الأخلاقي. تقوم بفحص كل طلب مقابل جميع السياسات النشطة، وتقرر ما إذا كان الطلب

يمكن أن يمر تلقائياً، أو يحتاج إلى مراجعة بشرية، أو يُرفض فوراً.

Python

```
from typing import Dict, Any, List, Tuple
import json
import time
import hashlib

class EthicalComplianceService:
    """
    خدمة الامتثال الأخلاقي - تفحص الطلبات ضد السياسات وتتخذ القرارات الأولية.
    Ethical Compliance Service - Screens requests against policies and makes initial decisions.
    """
    def __init__(self, db_path: str = "integrated_project.db"):
        self.db_path = db_path
        self.repo = EthicalRepository(db_path)

    def screen_request(self, request: SensitiveRequest) -> Dict[str, Any]:
        """
        فحص طلب ضد جميع السياسات النشطة.
        Screen a request against all active policies.
        """
        print(f"🔍 فحص الطلب: {request.request_type} (Target: {request.target_type}:
{request.target_id})")

        # 1. جلب جميع السياسات النشطة
        policies = self.repo.get_active_policies()
```

```
        results = {
            "request_id": request.request_id,
            "policies_checked": [],
            "overall_status": "pending",
            "requires_human_review": False,
            "compliance_log": []
        }

        # 2. فحص كل سياسة
        for policy in policies:
            check_result = self._evaluate_policy(policy, request)

            # تسجيل نتيجة الفحص
            result_enum = ComplianceResult.PASS if check_result["passed"] else (
                ComplianceResult.WARNING if check_result.get("warning", False) else
ComplianceResult.FAIL
            )

            self.repo.log_compliance_check(
                request_id=request.request_id,
                policy_id=policy.policy_id,
                result=result_enum,
                details=check_result.get("details", "")
            )

            results["policies_checked"].append({
                "policy_code": policy.policy_code,
                "policy_name": policy.policy_name,
                "passed": check_result["passed"],
                "details": check_result.get("details", "")
            })
```

```
        # 3. إذا فشلت السياسة، قرر الإجراء
        if not check_result["passed"]:
            # إذا كانت السياسة تتطلب مراجعة بشرية، قم بتعيين العلم
            if policy.requires_human_review:
                results["requires_human_review"] = True
                results["overall_status"] = "under_review"
                print(f"⚠️ السياسة {policy.policy_code} فشلت وتتطلب مراجعة بشرية")
            else:
                # إذا لم تتطلب مراجعة، ارتكب الفشل مباشرة
                results["overall_status"] = "rejected"
                results["rejection_reason"] = f"فشلت السياسة {policy.policy_code}:
{check_result.get('details', '')}"
                print(f"❌ تم رفض الطلب بسبب {policy.policy_code}")
                break
        else:
            # إذا كانت السياسة تتطلب مراجعة بشرية ولكنها اجتازت، نستمر في التقييم
            if policy.requires_human_review:
                results["requires_human_review"] = True

    # 4. إذا لم تفشل أي سياسة وكانت المراجعة البشرية مطلوبة، قم بتحديث الحالة
    if results["overall_status"] == "pending" and results["requires_human_review"]:
        results["overall_status"] = "under_review"
        print("📋 الطلب يحتاج إلى مراجعة بشرية (مرت جميع السياسات ولكن بعضها يتطلب مراجعة)")

    # 5. إذا لم تفشل أي سياسة ولا تتطلب مراجعة، الموافقة التلقائية
    if results["overall_status"] == "pending":
        results["overall_status"] = "approved"
        print("✅ تمت الموافقة التلقائية على الطلب")

    # 6. تحديث حالة الطلب في قاعدة البيانات
    self.repo.update_request_status(
        request.request_id,
```

```
        RequestStatus(results["overall_status"])
    )

    return results

def _evaluate_policy(self, policy: EthicalPolicy, request: SensitiveRequest) -> Dict[str, Any]:
    """
    تقييم سياسة واحدة ضد الطلب.
    Evaluate a single policy against the request.
    """
    # تطبيق القاعدة حسب نوع السياسة
    if policy.policy_type == "quantum_usage":
        # التحقق من استخدام الكم
        max_qubits = policy.rule_expression.get("max_qubits", 10)
        max_time = policy.rule_expression.get("max_time", 60)

        qubits = request.request_details.get("estimated_qubits", 0)
        time_sec = request.request_details.get("estimated_time_seconds", 0)

        if qubits > max_qubits:
            return {
                "passed": False,
                "details": f"عدد الكيوبتات ({qubits}) يتجاوز الحد المسموح ({max_qubits})"
            }
        if time_sec > max_time:
            return {
                "passed": False,
                "details": f"الزمن المقدر ({time_sec} ثانية) يتجاوز الحد المسموح ({max_time})"
            }
        return {"passed": True, "details": "الاستخدام الكمومي ضمن الحدود المسموحة"}

    elif policy.policy_type == "data_access":
```

```
    # التحقق من الوصول إلى البيانات
    allowed_types = policy.rule_expression.get("allowed_types", [])
    data_type = request.request_details.get("data_type", "unknown")
    if data_type in allowed_types:
        return {"passed": True, "details": f"نوع البيانات {data_type} مسموح به"}
    else:
        return {
            "passed": False,
            "details": f"نوع البيانات {data_type} غير مسموح به. الأنواع المسموحة: {allowed_types}"
        }

elif policy.policy_type == "gene_editing":
    # التحقق من التعديل الجيني
    edit_target = request.request_details.get("gene_symbol", "")
    allowed_genes = policy.rule_expression.get("allowed_genes", [])
    if not allowed_genes or edit_target in allowed_genes:
        return {"passed": True, "details": f"تعديل الجين {edit_target} مسموح به"}
    else:
        return {
            "passed": False,
            "details": f"تعديل الجين {edit_target} غير مسموح به"
        }

elif policy.policy_type == "human_review":
    # هذا النوع من السياسات دائما يتطلب مراجعة بشرية، ولكن لا يفشل
    return {
        "passed": True,
        "warning": True,
        "details": "هذه السياسة تتطلب مراجعة بشرية إضافية"
    }

# افتراضياً، تمر السياسة
```

```
return {"passed": True, "details": "السياسة غير محددة، تمر افتراضيا"}
```

الفصل الحادي والسبعون: خدمة المراجعة البشرية — سير العمل الكامل

Chapter Seventy-One: Human Review Service — Complete Workflow

الشكل 72 — الخريطة المفاهيمية للفصل

Figure 72 — Conceptual map of the chapter

من منظور المسار الكامل للكتاب، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

الشكل 72 — سير العمل الهندسي للفصل

Figure 72 — Engineering workflow of the chapter

The concepts developed in Part Two take on a new role here, you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

هذه الخدمة تدير عملية المراجعة البشرية من البداية إلى النهاية، بما في ذلك تعيين المراجعين، وتجميع الأدلة، وتسجيل القرارات، وتحديث سلسلة الموافقات.

Python

```
from typing import List, Dict, Any, Optional


class HumanReviewService:
    """
    خدمة المراجعة البشرية - تدير عملية المراجعة الكاملة.
    Human Review Service - Manages the complete review process.
    """
    def __init__(self, db_path: str = "integrated_project.db"):
        self.db_path = db_path
        self.repo = EthicalRepository(db_path)
        self.compliance = EthicalComplianceService(db_path)

    def get_review_queue(self, reviewer_id: Optional[int] = None, limit: int = 20) -> List[Dict[str,
Any]]:
        """
        استرجاع قائمة انتظار المراجعة للمراجع المحدد.
        Retrieve the review queue for the specified reviewer.
        """
        with sqlite3.connect(self.db_path) as conn:
            conn.row_factory = sqlite3.Row
            cursor = conn.cursor()
```

```
        # جلب الطلبات التي تحتاج إلى مراجعة
        query = """
          SELECT
            sr.*,
            u.username AS requester_name,
            ep.policy_code,
            ep.policy_name,
            COUNT(er.review_id) AS review_count
          FROM sensitive_requests sr
          JOIN users u ON sr.requester_id = u.user_id
          LEFT JOIN ethical_policies ep ON ep.policy_id IN (
            SELECT policy_id FROM ethical_compliance_log
            WHERE request_id = sr.request_id
            AND check_result = 'warning'
          )
          LEFT JOIN ethical_reviews er ON sr.request_id = er.request_id
          WHERE sr.status IN ('pending', 'under_review')
          GROUP BY sr.request_id
          ORDER BY sr.created_at ASC
          LIMIT ?
        """
        cursor.execute(query, (limit,))
        rows = cursor.fetchall()
        return [dict(row) for row in rows]

    def conduct_review(self, request_id: int, reviewer_id: int,
                       decision: str, rationale: str) -> Dict[str, Any]:
        """
        إجراء مراجعة بشرية لطلب معين.
        Conduct a human review for a specific request.
        """
        # 1. التحقق من أن الطلب لا يزال بحاجة إلى مراجعة
```

```
    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.cursor()
        cursor.execute("""
            SELECT status FROM sensitive_requests
            WHERE request_id = ?
        """, (request_id,))
        row = cursor.fetchone()
        if not row:
            return {"error": "الطلب غير موجود"}
        if row[0] not in ['pending', 'under_review']:
            return {"error": f"الطلب في حالة {row[0]}، لا يمكن مراجعته الآن"}

    # 2. جلب سياسة المراجعة (أول سياسة تتطلب مراجعة بشرية)
    policies = self.repo.get_active_policies()
    review_policy = next((p for p in policies if p.requires_human_review), None)
    if not review_policy:
        return {"error": "لا توجد سياسة تتطلب مراجعة بشرية"}

    # 3. تسجيل المراجعة
    review = EthicalReview(
        request_id=request_id,
        reviewer_id=reviewer_id,
        policy_id=review_policy.policy_id,
        decision=ReviewDecision(decision),
        rationale=rationale
    )
    review_id = self.repo.record_review(review)

    # 4. تحديث حالة الطلب
    new_status = (
        RequestStatus.APPROVED if decision == 'approved' else
        RequestStatus.REJECTED if decision == 'rejected' else
```

```
        RequestStatus.ESCALATED
    )
    self.repo.update_request_status(request_id, new_status)

    # 5. إضافة سلسلة الموافقات إذا كانت الموافقة
    if decision == 'approved':
        with sqlite3.connect(self.db_path) as conn:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO approval_chain
                (request_id, approver_id, approval_level, approval_decision)
                VALUES (?, ?, ?, ?)
            """, (request_id, reviewer_id, 1, 'approved'))
            conn.commit()

    # 6. سجل التدقيق الأخلاقي
    self._log_ethical_decision(request_id, reviewer_id, decision, rationale)

    return {
        "review_id": review_id,
        "request_id": request_id,
        "decision": decision,
        "new_status": new_status.value,
        "message": f"تم {'الموافقة على' if decision == 'approved' else 'رفض'} الطلب بنجاح"
    }

def _log_ethical_decision(self, request_id: int, reviewer_id: int,
                          decision: str, rationale: str) -> None:
    """تسجيل القرار الأخلاقي في سجل التدقيق."""
    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.cursor()
        # إنشاء جدول سجل القرارات الأخلاقية إذا لم يكن موجودا
```

```
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS ethical_decision_log (
                log_id INTEGER PRIMARY KEY AUTOINCREMENT,
                request_id INTEGER NOT NULL,
                reviewer_id INTEGER NOT NULL,
                decision TEXT NOT NULL,
                rationale TEXT NOT NULL,
                logged_at TEXT DEFAULT CURRENT_TIMESTAMP,
                decision_hash TEXT
            )
        """)
        # حساب بصمة القرار
        import hashlib
        decision_hash = hashlib.sha256(
            f"{request_id}|{reviewer_id}|{decision}|{rationale}".encode()
        ).hexdigest()[:16]

        cursor.execute("""
            INSERT INTO ethical_decision_log
            (request_id, reviewer_id, decision, rationale, decision_hash)
            VALUES (?, ?, ?, ?, ?)
        """, (request_id, reviewer_id, decision, rationale, decision_hash))
        conn.commit()

def escalate_request(self, request_id: int, reviewer_id: int, rationale: str) -> Dict[str, Any]:
    """
    تصعيد طلب إلى مستوى أعلى من المراجعة (مثل لجنة أخلاقية).
    Escalate a request to a higher review level (e.g., ethics committee).
    """
    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.cursor()
        cursor.execute("""
```

```
        UPDATE sensitive_requests
        SET status = 'escalated'
        WHERE request_id = ?
    """, (request_id,))
    conn.commit()

    # تسجيل التصعيد
    cursor.execute("""
        INSERT INTO ethical_reviews
        (request_id, reviewer_id, policy_id, review_decision, review_rationale)
        VALUES (?, ?, ?, ?, ?)
    """, (request_id, reviewer_id, 0, 'escalated', rationale))
    conn.commit()

return {
    "request_id": request_id,
    "status": "escalated",
    "message": f"تم تصعيد الطلب إلى المستوى الأعلى للمراجعة: {rationale}"
}
```

الفصل الثاني والسبعون: الدمج الكامل — نظام الحوكمة الأخلاقية الكمومية

Chapter Seventy-Two: Full Integration — The Quantum Ethical Governance System

الشكل 73 — الخريطة المفاهيمية للفصل

Figure 73 — Conceptual map of the chapter

هذه الخطوة لا تبدأ من الصفر؛ بل تعتمد على ما سبق بناؤه، تعلمت أن الحساب يجب أن يقود إلى تفسير قابل للتحقق؛ الآن ستطبق المبدأ نفسه عند الانتقال إلى النماذج الكمومية.

الشكل 73 — سير العمل الهندسي للفصل

Figure 73 — Engineering workflow of the chapter

From the perspective of the book's complete progression, you learned that computation must lead to a verifiable interpretation; now you will apply the same principle when entering quantum models.

هذا الفصل يدمج جميع المكونات في نظام واحد قابل للتشغيل. سنقوم ببناء واجهة سطر أوامر (CLI) متكاملة للتعامل مع الطلبات الأخلاقية، والمراجعات، والتقارير.

Python

```
# !/usr/bin/env python3
# ================================================================
# quantum_ethical_cli.py
# واجهة سطر الأوامر لنظام الحوكمة الأخلاقية الكمومية
# ================================================================

import argparse
import json
import sys
from typing import List, Dict, Any

class QuantumEthicalCLI:
    """
    واجهة سطر الأوامر لنظام الحوكمة الأخلاقية الكمومية.
    """
    def __init__(self):
        self.parser = self._build_parser()
        self.repo = EthicalRepository()
        self.compliance = EthicalComplianceService()
        self.review_service = HumanReviewService()

    def _build_parser(self) -> argparse.ArgumentParser:
```

```
parser = argparse.ArgumentParser(
    prog="qe-cli",
    description="نظام الحوكمة الأخلاقية الكمومية - إدارة الطلبات والمراجعات"
)

subparsers = parser.add_subparsers(dest="command", help="الأوامر المتاحة")

 #  ---------------------------
 #  الأمر: submit (تقديم طلب)
 #  ---------------------------
submit_parser = subparsers.add_parser("submit", help="تقديم طلب جديد")
submit_parser.add_argument("--type", required=True,
                        choices=['quantum_execution', 'gene_edit', 'data_export'],
                        help="نوع الطلب")
submit_parser.add_argument("--target-id", type=int, required=True,
                        help="معرف الهدف")
submit_parser.add_argument("--target-type", required=True,
                        choices=['sequence', 'sample', 'model'],
                        help="نوع الهدف")
submit_parser.add_argument("--requester-id", type=int, required=True,
                        help="معرف الطالب")
submit_parser.add_argument("--purpose", required=True,
                        help="الغرض من الطلب")
submit_parser.add_argument("--details", required=True,
                        help="تفاصيل الطلب (JSON)")

 #  ---------------------------
 #  الأمر: review (المراجعة)
 #  ---------------------------
review_parser = subparsers.add_parser("review", help="إدارة المراجعات")
review_subparsers = review_parser.add_subparsers(dest="review_action")
```

```
review_list = review_subparsers.add_parser("list", help="عرض قائمة المراجعة")
review_list.add_argument("--reviewer-id", type=int, help="معرف المراجع")
review_list.add_argument("--limit", type=int, default=20, help="عدد العناصر")

review_conduct = review_subparsers.add_parser("conduct", help="إجراء مراجعة")
review_conduct.add_argument("--request-id", type=int, required=True,
                help="معرف الطلب")
review_conduct.add_argument("--reviewer-id", type=int, required=True,
                help="معرف المراجع")
review_conduct.add_argument("--decision", required=True,
                choices=['approved', 'rejected', 'escalated'],
                help="قرار المراجعة")
review_conduct.add_argument("--rationale", required=True,
                help="مبرر القرار")

 # ---------------------------
 # الأمر: audit (سجل التدقيق)
 # ---------------------------
audit_parser = subparsers.add_parser("audit", help="عرض سجل التدقيق الأخلاقي")
audit_parser.add_argument("--request-id", type=int, help="فلترة حسب الطلب")
audit_parser.add_argument("--limit", type=int, default=50, help="عدد السجلات")

 # ---------------------------
 # الأمر: policy (إدارة السياسات)
 # ---------------------------
policy_parser = subparsers.add_parser("policy", help="إدارة السياسات الأخلاقية")
policy_subparsers = policy_parser.add_subparsers(dest="policy_action")

policy_list = policy_subparsers.add_parser("list", help="عرض السياسات النشطة")
policy_create = policy_subparsers.add_parser("create", help="إنشاء سياسة جديدة")
policy_create.add_argument("--code", required=True, help="رمز السياسة")
policy_create.add_argument("--name", required=True, help="اسم السياسة")
```

```
        policy_create.add_argument("--type", required=True,
                              choices=['quantum_usage', 'data_access', 'gene_editing', 'human_review'],
                              help="نوع السياسة")
        policy_create.add_argument("--rule", required=True, help="قاعدة السياسة (JSON)")
        policy_create.add_argument("--risk", default="medium",
                              choices=['low', 'medium', 'high'], help="مستوى المخاطر")
        policy_create.add_argument("--requires-review", action='store_true',
                              help="يتطلب مراجعة بشرية")

        return parser

    def run(self, args: List[str] = None) -> None:
        parsed = self.parser.parse_args(args)

        if not parsed.command:
            self.parser.print_help()
            return

        if parsed.command == "submit":
            self._handle_submit(parsed)
        elif parsed.command == "review":
            self._handle_review(parsed)
        elif parsed.command == "audit":
            self._handle_audit(parsed)
        elif parsed.command == "policy":
            self._handle_policy(parsed)
        else:
            print(f"أمر غير معروف: {parsed.command}")

    # ----------------------------
    # معالجات الأوامر
    # ----------------------------
```

```
def _handle_submit(self, args) -> None:
    """تقديم طلب جديد."""
    try:
        details = json.loads(args.details)
    except json.JSONDecodeError:
        print("❌ تنسيق JSON غير صحيح")
        return

    request = SensitiveRequest(
        request_type=args.type,
        target_id=args.target_id,
        target_type=args.target_type,
        requester_id=args.requester_id,
        request_purpose=args.purpose,
        request_details=details
    )

    request_id = self.repo.submit_request(request)
    print(f"✅ تم تقديم الطلب بنجاح. المعرف: {request_id}")

    # تشغيل الفحص التلقائي
    print("🔍 جاري الفحص التلقائي...")
    request.request_id = request_id
    result = self.compliance.screen_request(request)

    print(f"📊 حالة الطلب: {result['overall_status']}")
    if result.get('rejection_reason'):
        print(f"❌ سبب الرفض: {result['rejection_reason']}")
    if result['requires_human_review']:
        print("📋 الطلب يحتاج إلى مراجعة بشرية")
```

```
    def _handle_review(self, args) -> None:
        """إدارة المراجعات."""
        if args.review_action == "list":
            queue = self.review_service.get_review_queue(
                reviewer_id=args.reviewer_id,
                limit=args.limit
            )
            print("=" * 80)
            print(f"📋 قائمة انتظار المراجعة ({len(queue)} عنصر)")
            print("=" * 80)
            for item in queue:
                print(f"معرف: {item['request_id']} | النوع: {item['request_type']} | الهدف:
{item['target_type']}:{item['target_id']}")
                print(f"  الطالب: {item['requester_name']} | الحالة: {item['status']}")
                print(f"  الغرض: {item['request_purpose']}")
                print("-" * 40)

        elif args.review_action == "conduct":
            result = self.review_service.conduct_review(
                request_id=args.request_id,
                reviewer_id=args.reviewer_id,
                decision=args.decision,
                rationale=args.rationale
            )
            if "error" in result:
                print(f"❌ {result['error']}")
            else:
                print(f"✅ {result['message']}")
                print(f"📊 الحالة الجديدة: {result['new_status']}")

        else:
            print("استخدم: qe-cli review {list|conduct} ...")
```

```
def _handle_audit(self, args) -> None:
    """عرض سجل التدقيق."""
    with sqlite3.connect("integrated_project.db") as conn:
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()

        query = """
            SELECT * FROM ethical_decision_log
            WHERE 1=1
        """
        params = []
        if args.request_id:
            query += " AND request_id = ?"
            params.append(args.request_id)
        query += " ORDER BY logged_at DESC LIMIT ?"
        params.append(args.limit)

        cursor.execute(query, params)
        rows = cursor.fetchall()

        print("=" * 80)
        print("📜 سجل القرارات الأخلاقية")
        print("=" * 80)
        for row in rows:
            print(f"معرف الطلب: {row['request_id']} | القرار: {row['decision']}")
            print(f"  المراجع: {row['reviewer_id']} | التوقيت: {row['logged_at']}")
            print(f"  المبرر: {row['rationale'][:100]}...")
            print("-" * 40)

def _handle_policy(self, args) -> None:
    """إدارة السياسات الأخلاقية."""
```

```
if args.policy_action == "list":
    policies = self.repo.get_active_policies()
    print("=" * 80)
    print("📋 السياسات الأخلاقية النشطة")
    print("=" * 80)
    for p in policies:
        print(f"{p.policy_code}: {p.policy_name} ({p.policy_type})")
        print(f"  المخاطر: {p.risk_level} | مراجعة بشرية: {'نعم' if p.requires_human_review else 'لا'}")
        print(f"  القاعدة: {json.dumps(p.rule_expression, indent=2)[:100]}...")
        print("-" * 40)

elif args.policy_action == "create":
    try:
        rule = json.loads(args.rule)
    except json.JSONDecodeError:
        print("❌ تنسيق JSON غير صحيح")
        return

    policy = EthicalPolicy(
        policy_code=args.code,
        policy_name=args.name,
        policy_type=args.type,
        rule_expression=rule,
        risk_level=args.risk,
        requires_human_review=args.requires_review,
        created_by="cli_user"
    )
    policy_id = self.repo.create_policy(policy)
    print(f"✅ تم إنشاء السياسة {args.code} بنجاح. المعرف: {policy_id}")

else:
    print("استخدم: qe-cli policy {list|create} ...")
```

```
def main():
    cli = QuantumEthicalCLI()
    cli.run(sys.argv[1:])

if __name__ == "__main__":
    main()
```

الفصل الثالث والسبعون: اختبار النظام الأخلاقي — سيناريو واقعي

Chapter Seventy-Three: Testing the Ethical System — A Realistic Scenario

الاختبار الأخلاقي الواقعي لا يبحث فقط عن المسار الصحيح؛ بل يحاول كسر النظام عمدًا. يجب أن تسأل: ماذا يحدث إذا كانت provenance ناقصة؟ ماذا لو وصل سجل بإصدار قديم؟ ماذا لو كانت النتيجة الكمومية مختلفة عن baseline؟ ماذا لو طلب مستخدم غير مخول بيانات حساسة؟ وماذا لو فشل أحد مكونات المسار في منتصف التنفيذ؟ النظام الجيد لا يخفي هذه الحالات؛ بل يحولها إلى حالات فشل واضحة، قابلة للتسجيل والمراجعة والاسترداد.

A realistic ethical test does not look only for the correct path; it deliberately tries to break the system. What happens when provenance is missing, when an old data version is submitted, when a quantum result disagrees with the baseline, when an unauthorized user requests sensitive data, or when a workflow component fails halfway through execution? A robust system does not hide these conditions. It turns them into explicit failure states that can be logged, reviewed, and recovered.

الشكل 74 — الخريطة المفاهيمية للفصل

Figure 74 — Conceptual map of the chapter

في الموضع الحالي من الرحلة التعليمية، بنيت أساس Python وBiopython والتحليل القابل لإعادة الإنتاج؛ الآن ستوسع هذا الأساس إلى طبقة هندسية جديدة.

الشكل 74 — سير العمل الهندسي للفصل

Figure 74 — Engineering workflow of the chapter

This step does not start from scratch; it builds on what came before, you built a foundation in Python, Biopython, and reproducible analysis; now you will extend that foundation into a new engineering layer.

هذا الفصل يقدم سيناريو اختبار كامل للنظام الأخلاقي، مع بيانات وهمية ولكنها تحاكي عمليات حقيقية.

Python

```
# ================================================================
# test_ethical_system.py
# اختبار نظام الحوكمة الأخلاقية الكمومية
# ================================================================

import json
import time
from integrated_project.ethical import *

def create_test_user():
    """إنشاء مستخدم اختبار."""
    with sqlite3.connect("integrated_project.db") as conn:
```

```
        cursor = conn.cursor()
        # إنشاء جدول المستخدمين إذا لم يكن موجودا
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS users (
                user_id INTEGER PRIMARY KEY AUTOINCREMENT,
                username TEXT UNIQUE NOT NULL,
                role TEXT,
                created_at TEXT DEFAULT CURRENT_TIMESTAMP
            )
        """)
        # إضافة مستخدمين للاختبار
        cursor.executemany("""
            INSERT OR IGNORE INTO users (user_id, username, role)
            VALUES (?, ?, ?)
        """, [
            (1, 'researcher_01', 'researcher'),
            (2, 'reviewer_01', 'reviewer'),
            (3, 'admin_01', 'admin')
        ])
        conn.commit()
    print("✅ تم إنشاء مستخدمين للاختبار")

def create_test_policies():
    """إنشاء سياسات اختبار."""
    repo = EthicalRepository()
    policies = [
        EthicalPolicy(
            policy_code="ETH-001",
            policy_name="الاستخدام الكمومي الآمن",
            policy_type="quantum_usage",
            rule_expression={"max_qubits": 5, "max_time": 30},
            risk_level="medium",
```

```
        requires_human_review=True,
        created_by="admin"
    ),
    EthicalPolicy(
        policy_code="ETH-002",
        policy_name="الوصول إلى بيانات المرضى",
        policy_type="data_access",
        rule_expression={"allowed_types": ["genomic_sequence", "clinical_metadata"]},
        risk_level="high",
        requires_human_review=True,
        created_by="admin"
    ),
    EthicalPolicy(
        policy_code="ETH-003",
        policy_name="تعديل الجينات المسموح بها",
        policy_type="gene_editing",
        rule_expression={"allowed_genes": ["BRCA1", "TP53", "KRAS"]},
        risk_level="high",
        requires_human_review=True,
        created_by="admin"
    ),
    EthicalPolicy(
        policy_code="ETH-004",
        policy_name="المراجعة البشرية العامة",
        policy_type="human_review",
        rule_expression={"requires_review": True},
        risk_level="low",
        requires_human_review=True,
        created_by="admin"
    )
]
```

```
    for policy in policies:
        repo.create_policy(policy)
    print("✅ تم إنشاء السياسات الاختبارية")

def run_test_scenario():
    """تشغيل سيناريو اختبار كامل."""
    print("=" * 80)
    print("🧪 سيناريو اختبار نظام الحوكمة الأخلاقية الكمومية")
    print("=" * 80)

    # 1. تهيئة النظام
    create_test_user()
    create_test_policies()

    repo = EthicalRepository()
    compliance = EthicalComplianceService()
    review_service = HumanReviewService()

    # 2. سيناريو 1: طلب كمومي صحيح (يجب أن يمر)
    print("\n📌 سيناريو 1: طلب كمومي ضمن الحدود المسموحة")
    request1 = SensitiveRequest(
        request_type="quantum_execution",
        target_id=100,
        target_type="sequence",
        requester_id=1,
        request_purpose="تحليل القيم الذاتية لجين BRCA1",
        request_details={
            "estimated_qubits": 3,
            "estimated_time_seconds": 10,
            "data_type": "genomic_sequence",
            "gene_symbol": "BRCA1"
        }
```

```
)
request1_id = repo.submit_request(request1)
print(f"✅ تم تقديم الطلب 1: {request1_id}")

result1 = compliance.screen_request(request1)
print(f"📊 حالة الطلب 1: {result1['overall_status']}")

# 3. سيناريو 2: طلب كمومي خارج الحدود (يجب أن يرفض)
print("\n📌 سيناريو 2: طلب كمومي خارج الحدود المسموحة")
request2 = SensitiveRequest(
    request_type="quantum_execution",
    target_id=101,
    target_type="sequence",
    requester_id=1,
    request_purpose="تحليل جينوم كامل",
    request_details={
        "estimated_qubits": 15,  # تجاوز الحد
        "estimated_time_seconds": 100,
        "data_type": "genomic_sequence",
        "gene_symbol": "BRCA1"
    }
)
request2_id = repo.submit_request(request2)
print(f"✅ تم تقديم الطلب 2: {request2_id}")

result2 = compliance.screen_request(request2)
print(f"📊 حالة الطلب 2: {result2['overall_status']}")
if result2.get('rejection_reason'):
    print(f"❌ سبب الرفض: {result2['rejection_reason']}")

# 4. سيناريو 3: طلب تعديل جيني غير مسموح (يجب أن يرفض أو يحتاج مراجعة)
print("\n📌 سيناريو 3: طلب تعديل جيني غير مسموح")
```

```
request3 = SensitiveRequest(
    request_type="gene_edit",
    target_id=102,
    target_type="sequence",
    requester_id=1,
    request_purpose="تعديل جين MYC",
    request_details={
        "gene_symbol": "MYC",  # غير مدرج في السياسة
        "edit_type": "knockout",
        "data_type": "genomic_sequence"
    }
)
request3_id = repo.submit_request(request3)
print(f"✅ تم تقديم الطلب 3: {request3_id}")

result3 = compliance.screen_request(request3)
print(f"📊 حالة الطلب 3: {result3['overall_status']}")

 # 5. سيناريو 4: طلب يحتاج إلى مراجعة بشرية (تمرير المراجعة)
print("\n📌 سيناريو 4: طلب يحتاج إلى مراجعة بشرية - الموافقة")
request4 = SensitiveRequest(
    request_type="data_export",
    target_id=103,
    target_type="sample",
    requester_id=1,
    request_purpose="تصدير بيانات سريرية للبحث",
    request_details={
        "data_type": "clinical_metadata",
        "export_format": "csv",
        "fields": ["age", "diagnosis", "treatment"]
    }
)
```

```
    request4_id = repo.submit_request(request4)
    print(f"✅ تم تقديم الطلب 4: {request4_id}")

    result4 = compliance.screen_request(request4)
    print(f"📊 حالة الطلب 4: {result4['overall_status']}")

    if result4['overall_status'] == 'under_review':
        print("📋 الطلب في قائمة المراجعة، جاري المراجعة...")
        # عرض قائمة المراجعة
        queue = review_service.get_review_queue(limit=5)
        print(f"📋 قائمة المراجعة: {len(queue)} عنصر")

        # إجراء مراجعة
        review_result = review_service.conduct_review(
            request_id=request4_id,
            reviewer_id=2,
            decision='approved',
            rationale='تم التحقق من الغرض العلمي وموافقة المريض. الطلب مصرح به.'
        )
        print(f"✅ نتيجة المراجعة: {review_result['message']}")

    # 6. تقرير الملخص النهائي
    print("\n" + "=" * 80)
    print("📊 تقرير ملخص الاختبار")
    print("=" * 80)
    print(f"الطلب 1 (كمومي ضمن الحدود): {result1['overall_status']}")
    print(f"الطلب 2 (كمومي خارج الحدود): {result2['overall_status']}")
    print(f"الطلب 3 (تعديل جيني غير مسموح): {result3['overall_status']}")
    print(f"الطلب 4 (مراجعة بشرية): {result4['overall_status']} → تمت المراجعة")

if __name__ == "__main__":
    run_test_scenario()
```

الخاتمة الهندسية —— ربط الأخلاقيات بالهندسة

Engineering Conclusion — Connecting Ethics to Engineering.

ما أنجزناه في هذا الباب:

1. قاعدة بيانات أخلاقية كاملة: 7 جداول مترابطة تخزن السياسات، الطلبات، المراجعات، وسجلات الامتثال.
2. طبقة OOP متكاملة: كائنات، مستودعات، وخدمات للتعامل مع كل جانب من جوانب الحوكمة الأخلاقية.
3. نظام امتثال تلقائي: يفحص كل طلب ضد السياسات النشطة ويقرر مصيره (موافقة، رفض، مراجعة).
4. خدمة مراجعة بشرية: تدير قائمة انتظار المراجعة، وتسجل القرارات، وتحديث السلاسل الزمنية للموافقات.

5. واجهة سطر أوامر: للتفاعل مع النظام (تقديم طلبات، إدارة مراجعات، عرض سجلات).

6. سيناريو اختبار: يحاكي أربع حالات مختلفة ويوضح كيفية عمل النظام في الممارسة.

What we accomplished in this unit:

1. Complete ethical database: 7 interconnected tables storing policies, requests, reviews, and compliance logs.
2. Integrated OOP layer: Objects, repositories, and services to handle every aspect of ethical governance.
3. Automated compliance system: Screens every request against active policies and determines its fate (approval, rejection, review).
4. Human review service: Manages the review queue, records decisions, and updates approval chains.
5. Command-line interface: For interacting with the system (submitting requests, managing reviews, viewing logs).
6. Test scenario: Simulates four different cases and demonstrates how the system works in practice.

الدرس النهائي:

The Final Lesson:

الأخلاقيات ليست خطبة تُلقى في نهاية المشروع. هي متطلب هندسي يُبنى في قلب النظام، تماماً مثل قواعد البيانات والواجهات البرمجية. عندما تُصمم الأخلاقيات ككود، تصبح قابلة للاختبار، والتوسع، والمراجعة، والتحسين. هذا هو جوهر "هندسة الحياة" التي بدأناها مع الحروف الأربعة وانتهينا بها مع الكيوبت والضمير.

Ethics is not a speech delivered at the end of the project. It is an engineering requirement built into the heart of the system, just like databases and APIs. When ethics is designed as code, it becomes testable, extensible, reviewable, and improvable. This is the essence of "Engineering Life" that we began with the four letters and concluded with the qubit and conscience.

الأساس العلمي لمؤثرات الكثافة والأنظمة الكمومية المفتوحة والقياس: Quantum control theory and applications، arXiv:0910.2350.

Scientific basis for density operators, open quantum systems, and measurement: Quantum control theory and applications, arXiv:0910.2350.

Quantum state tomography and measurement complexity: Gross et al., Quantum state tomography via compressed sensing, arXiv:0909.3304.

التصوير المقطعي للحالة الكمومية وتعقيد القياس: Gross وآخرون، Quantum state tomography via compressed sensing، arXiv:0909.3304.

VQE and Hamiltonian ground-state estimation: Tilly et al., The Variational Quantum Eigensolver: a review of methods and best practices, arXiv:2111.05176.

VQE وتقدير الحالة الأرضية للهاميلتوني: Tilly وآخرون، The Variational Quantum Eigensolver: a review of methods and best practices، arXiv:2111.05176.

NISQ algorithms, limitations, and resource constraints: Bharti et al., Noisy intermediate-scale quantum (NISQ) algorithms, arXiv:2101.08448.

خوارزميات NISQ وحدودها وقيود الموارد: Bharti وآخرون، Noisy intermediate-scale quantum (NISQ) algorithms، arXiv:2101.08448.

Quantum error mitigation: Cai et al., Quantum Error Mitigation, arXiv:2210.00921.

تخفيف الأخطاء الكمومية: Cai وآخرون، Quantum Error Mitigation، arXiv:2210.00921.

Hybrid workflow performance and classical–quantum coordination: Rao et al., Performance Model for Hybrid Quantum-Classical Workflows, arXiv:2607.15426.

أداء سير العمل الهجين والتنسيق بين الحوسبة الكلاسيكية والكمومية: Rao وآخرون، Performance Model for Hybrid Quantum-Classical Workflows، arXiv:2607.15426.

IBM Quantum Learning — quantum measurement and circuits: https://quantum.cloud.ibm.com/learning/

IBM Quantum Learning — القياس الكمومي والدوائر: https://quantum.cloud.ibm.com/learning/

Scientific and Technical References.

Python Software Foundation. sqlite3 documentation. https://docs.python.org/3/library/sqlite3.html.

SQLite. CREATE INDEX documentation. https://www.sqlite.org/lang_createindex.html.

Docker Docs. Dockerfile reference and build documentation. https://docs.docker.com/reference/dockerfile.

Snakemake Documentation. Snakefiles and Rules. https://snakemake.readthedocs.io/en/stable/snakefiles/rules.html.

Python Packaging User Guide. Building and Publishing. https://packaging.python.org/en/latest/guides/section-build-and-publish/

pytest Documentation. https://docs.pytest.org/en/stable/

Streamlit Documentation. Input widgets. https://docs.streamlit.io/develop/api-reference/widgets.

Dask Documentation. dask.dataframe.read_csv and DataFrame partitioning. https://docs.dask.org/en/stable/

Ensembl Documentation. API and relational database resources. https://www.ensembl.org/info/docs/api/index.html.

UCSC Genome Browser. MySQL/MariaDB access documentation. https://www.genome.ucsc.edu/goldenPath/help/mysql.html.

Biopython Tutorial and Cookbook. https://biopython.org/docs/latest/Tutorial/

Cock, P. J. A., et al. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics, 25(11), 1422–1423.

Jinek, M., et al. (2012). A programmable dual-RNA-guided DNA endonuclease in adaptive bacterial immunity. Science, 337(6096), 816–821.

Doudna, J. A., & Charpentier, E. (2014). The new frontier of genome engineering with CRISPR-Cas9. Science, 346(6213), 1258096.

Hsu, P. D., Lander, E. S., & Zhang, F. (2014). Development and applications of CRISPR-Cas9 for genome engineering. Cell, 157(6), 1262–1278.

Scientific-integrity note: this edition expands Part Three using the Part Two table of contents supplied in the conversation, the expanded Part Three file available here, and verification of technical properties against official documentation. It does not claim to reconstruct unavailable Part Two text verbatim.

المراجع العلمية والتقنية

ملاحظة عن الأمانة العلمية: هذه النسخة توسّع الجزء الثالث اعتماداً على الفهرس الذي قدمته، وعلى النسخة الموسعة المتاحة في المحادثة، وعلى التحقق من الخصائص التقنية بمراجع رسمية. لا تدّعي أنها إعادة بناء حرفية لنص الجزء الثاني غير المتاح كملف كامل في هذه اللحظة.

1. Lesk, A. M. (2019). Introduction to Bioinformatics (5th ed.). Oxford University Press.

2. Bourne, P. E., & Weissig, H. (Eds.). (2003). Structural Bioinformatics. Wiley-Liss.

3. Buneman, P., et al. (2008). "Why and Where: A Characterization of Data Provenance." In Proceedings of the 8th International Conference on Database Theory.

4. ECMA-404. The JSON Data Interchange Format. Ecma International.

5. RFC 4180. Common Format and MIME Type for CSV Files. Internet Engineering Task Force (IETF).

6. Chen, P. P. (1976). "The Entity-Relationship Model—Toward a Unified View of Data." ACM Transactions on Database Systems, 1(1), 9–36.

7. Elmasri, R., & Navathe, S. B. (2016). Fundamentals of Database Systems (7th ed.). Pearson.

8. Teorey, T. J., Lightstone, S. S., & Nadeau, T. (2011). Database Modeling and Design: Logical Design (5th ed.). Morgan Kaufmann.

9. Date, C. J. (2019). An Introduction to Database Systems (8th ed.). Addison-Wesley.

10. Beaulieu, A. (2020). Learning SQL (3rd ed.). O'Reilly Media.

11. SQLite Documentation. (2025). SQLite Official Documentation. https://www.sqlite.org/docs.html.

12. Mungall, C. J., Emmert, D. B., & FlyBase Consortium. (2007). "A Chado case study: an ontology-based modular schema for representing genome-associated biological information." Bioinformatics, 23(13), i337–i346.

13. BioSQL Schema. (2025). BioSQL Official Documentation. https://biosql.org/

14. Codd, E. F. (1970). "A Relational Model of Data for Large Shared Data Banks." Communications of the ACM, 13(6), 377–387.

15. Kent, W. (1983). "A Simple Guide to Five Normal Forms in Relational Database Theory." Communications of the ACM, 26(2), 120–125.

16. SQLAlchemy Documentation. (2025). SQLAlchemy Official Documentation. https://docs.sqlalchemy.org/

17. Myers, J., & Copeland, R. (2016). Essential SQLAlchemy (2nd ed.). O'Reilly Media.

18. Biopython's BioSQL Module. (2025). Biopython Official Documentation. https://biopython.org/docs/latest/api/BioSQL.html.

19. Ramakrishnan, R., & Gehrke, J. (2003). Database Management Systems (3rd ed.). McGraw-Hill.

20. Python's sqlite3 module. (2025). Python Official Documentation. https://docs.python.org/3/library/sqlite3.html.

21. Graefe, G. (1993). "Query Evaluation Techniques for Large Databases." ACM Computing Surveys, 25(2), 73–169.

22. PostgreSQL Documentation: "Performance Tips". (2025). PostgreSQL Official Documentation. https://www.postgresql.org/docs/current/performance-tips.html.
23. Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley.
24. Altman, D. G. (1991). Practical Statistics for Medical Research. Chapman & Hall.
25. Rosner, B. (2015). Fundamentals of Biostatistics (8th ed.). Cengage Learning.
26. Strang, G. (2016). Introduction to Linear Algebra (5th ed.). Wellesley-Cambridge Press.
27. Lay, D. C., Lay, S. R., & McDonald, J. J. (2015). Linear Algebra and Its Applications (5th ed.). Pearson.
28. Trefethen, L. N., & Bau, D. (1997). Numerical Linear Algebra. Society for Industrial and Applied Mathematics (SIAM).
29. Golub, G. H., & Van Loan, C. F. (2013). Matrix Computations (4th ed.). Johns Hopkins University Press.
30. Demmel, J. W. (1997). Applied Numerical Linear Algebra. Society for Industrial and Applied Mathematics (SIAM).
31. Kolda, T. G., & Bader, B. W. (2009). "Tensor Decompositions and Applications." SIAM Review, 51(3), 455–500.

32. Nielsen, M. A., & Chuang, I. L. (2010). Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press.

33. Mermin, N. D. (2007). Quantum Computer Science: An Introduction. Cambridge University Press.

34. IBM Quantum Documentation. (2025). IBM Quantum Official Documentation. https://docs.quantum.ibm.com/

35. Nałęcz-Charkiewicz, K., Charkiewicz, K., & Nowak, R. M. (2024). "Quantum computing in bioinformatics: a systematic review mapping." Briefings in Bioinformatics, 25(5).

36. Nassir, N., et al. (2025). "Quantum computing and the implementation of precision medicine." npj Genomic Medicine, 10(1), 1.

37. Mokhtari, M., et al. (2024). "New classifications for quantum bioinformatics." Briefings in Bioinformatics, 25(2).

38. McClean, J. R., et al. (2016). "The theory of variational hybrid quantum-classical algorithms." New Journal of Physics, 18(2), 023023.

39. Benson, D. A., et al. (2024). "GenBank 2024 Update." Nucleic Acids Research, 52(D1), D134–D137.

40. Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

41. Pérez-Riverol, Y., et al. (2025). "Omilayers: a Python package for efficient data management to support multi-omic analysis." BMC Bioinformatics.

42. NCBI Bookshelf. (2025). National Center for Biotechnology Information. https://www.ncbi.nlm.nih.gov/books/

43. Entrez Help. (2025). NCBI Entrez Utilities Documentation. https://www.ncbi.nlm.nih.gov/books/NBK25501/

44. SQLite Official Website. (2025). https://sqlite.org/

45. Qiskit. (2025). IBM Quantum SDK Documentation. https://qiskit.org/

46. Biopython Project. (2026). Biopython: Freely available Python tools for computational molecular biology. https://biopython.org/

47. gget. (2026). gget: Efficient querying of genomic reference databases. GitHub. https://github.com/scverse/gget .

48. PostgreSQL Global Development Group. (2025). PostgreSQL: The World's Most Advanced Open Source Relational Database. https://www.postgresql.org/

49. MySQL. (2025). MySQL: Open Source Relational Database Management System. https://www.mysql.com/

50. SQLite. (2025). SQLite: Self-Contained, Serverless, Zero-Configuration, Transactional SQL Database Engine. https://sqlite.org/

51. Qiskit. (2025). Qiskit: Open-source Quantum Computing Framework. IBM Quantum. https://qiskit.org/

52. Cirq. (2025). Cirq: Google's Open-source Quantum Computing Framework. https://quantumai.google/cirq .

53. IBM/QBioCode. (2026). QBioCode: Computational resources for quantum applications in healthcare and life science. GitHub. https://github.com/IBM/QBioCode .

54. QuASeR. (2025). Quantum Accelerated de novo DNA Sequence Reconstruction. GitHub. https://github.com/QE-Lab/QuASeR .

55. qbiocode. (2026). qbiocode: Quantum computing applications in healthcare and life science data analysis. PyPI. https://pypi.org/project/qbiocode/

56. RnaFoldQuantum. (2025). Quantum-Enhanced RNA Therapeutic Design System. GitHub. https://github.com/jshaigler/RnaFoldQuantum .

57. InMoose. (2025). Integrated Multi-Omic Open-Source Environment in Python. GitHub. https://github.com/epigenelabs/inmoose .

58. LaminDB. (2025). Open-source data framework for biology: queryable, traceable, reproducible, FAIR data. https://lamin.ai/

59. BioCypher. (2025). Open-source ecosystem for building FAIR biomedical knowledge graphs. https://biocypher.org/

60. BioDWH2. (2024). Open-source graph-based data warehouse and mapping tool. GitHub. https://github.com/BioDWH2 .

61. KBase. (2026). Open collaborative platform for biological data analysis. https://www.kbase.us/

62. pyBiodatafuse. (2025). Open-source tool for modular queries across biomedical resources. GitHub. https://github.com/BioDataFuse/pyBiodatafuse .

63. Bio4j. (2025). Open-source graph-based bioinformatics data platform integrating open data sources. https://bio4j.com/

64. UniProt. (2025). Universal Protein Resource: Freely accessible protein sequence and functional information. https://www.uniprot.org/

65. GenBank. (2025). NIH Genetic Sequence Database. NCBI. https://www.ncbi.nlm.nih.gov/genbank/

66. Protein Data Bank (PDB). (2025). Global repository for 3D structural data of biological macromolecules. https://www.wwpdb.org/

67. NCBI Entrez Programming Utilities (E-utilities). (2025). API for programmatic access to NCBI databases. https://www.ncbi.nlm.nih.gov/books/NBK25501.

68. BioSQL. (2025). Open-source biological sequence database schema. https://